# A New Keystream Generator MUGI

Dai Watanabe[1], Soichi Furuya[1], Hirotaka Yoshida[1], Kazuo Takaragi[1], and Bart Preneel[2]

[1] Systems Development Laboratory, Hitachi, Ltd., 292 Yoshida-cho, Totsuka-ku, Yokohama, 244-0817, Japan
{daidai, soichi, takara}@sdl.hitachi.co.jp
[2] Katholieke Universiteit Leuven, Dept. Electrical Engineering-ESAT, Kasteelpark Arenberg 10, B-3001 Heverlee, Belgium
Bart.Preneel@esat.kuleuven.ac.be

**Abstract.** We present a new keystream generator (KSG) MUGI, which is a variant of Panama proposed at FSE '98. MUGI has a 128-bit secret key and a 128-bit initial vector as parameters and generates a 64-bit string per round. The design is particularly suited for efficient hardware implementations, but the software performance of MUGI is excellent as well. A speed optimized implementation in hardware achieves about 3 Gbps with 26 Kgates, which is several times faster than AES. On the other hand the security was evaluated according to re-synchronization attack, related-key attack, and linear correlation of an output sequence. Our analysis confirms that MUGI is a secure KSG.

**Keywords.** Keystream generator, Block cipher, Panama, Re-synchronization attack, Related-key attack.

## 1 Introduction

This paper presents a new keystream generator MUGI that is designed for use as a stream cipher. MUGI has a 256-bit input (consisting of a 128-bit secret key and a 128-bit public parameter IV) and outputs a 64-bit random data block for each round.

Several approaches are known in the literature to the design of KSGs. One particularly popular approach is based on Linear Feedback Shift Registers (LFSRs). They are suitable for very compact hardware implementations and provide good randomness. However, due to their linearity and predictability, they cannot be used in their pure forms. Several techniques have been developed to improve their security, such as the combination generator, non-linear filtering, and clock control. A substantial amount of research has been spent on the security of these schemes. But LFSRs are not suited for efficient software implementations.

On the other hand software-oriented stream ciphers seem to be designed in an *ad hoc* way, and we do not seem to have the appropriate tools to evaluate them. The most important criterion is to verify deviations from randomness. Examples in this class include (Alleged) RC4 [Sc96] (security analysis in [FS01]), SEAL 3.0

[RC98] (security analysis in [Fl01]), LEVIATHAN [McF00] (security analysis in [CL01]), and LILI-128 [CGMPS00] (security analysis in [MFI01]).

In this paper we focus on PANAMA, designed by J. Daemen and C. Clapp [DC98]. PANAMA is based on generic design principles, comparable to those of block ciphers. PANAMA can be used both as a KSG and as a hash function. However, recently Rijmen *et al.* [RRPV01] have exposed security weaknesses in the security of PANAMA as a hash function. But these weaknesses have no impact at all on the security of PANAMA as a KSG.

MUGI is a variant of PANAMA which is only suitable as a KSG. The design goal is to make MUGI suitable for many platforms. As a result, MUGI achieves a performance that is equal to or even better than AES [DR99], especially the hardware performance is excellent. MUGI can be implemented in hardware with 18 Kgates. In terms of security, we evaluate the security against re-synchronization attacks [DGV94] and related-key attacks. Furthermore we calculate the linear correlation of the output sequence. As the result, we conclude that MUGI is a reliable and efficient cryptographic primitive that can be used to provide encryption and message authentication.

This article is organized as follows: in Sect. 2 we describe the generalization of a PANAMA-like structure and discuss the security of this type of KSG. The specification of MUGI is given in Sect. 3. In Sect. 4 we present some results about the security of MUGI. In Sect. 5 we discuss the implementation of MUGI both in software and hardware. In Appendix, we give test vectors of MUGI. You can find the perfect version of this paper at `http://www.sdl.hitachi.co.jp/crypto/mugi/index-e.html`.

## 2 Design Policy

### 2.1 PANAMA-Like Keystream Generator

The principal part of a KSG is a set $(\mathcal{S}, \Upsilon, f)$ which consists of an internal state $\mathcal{S}$, its update function $\Upsilon$, and the output filter $f$ which abstracts the output sequence from the internal state $\mathcal{S}$. We call the set $(\mathcal{S}, \Upsilon)$ the *internal-state machine*. In addition we call a single application of the state update function a *round*. $\mathcal{S}^{(t)}$ refers to the internal state at round $t$.

For PANAMA, the internal state is divided into two parts, state $a$ and buffer $b$. The update function of PANAMA depends in a different way on different parts of the internal state. Note that each update function uses another internal state as a parameter. We denote the update functions of state $a$ and buffer $b$ with $\rho$ and $\lambda$, respectively.

The noteworthy characteristic of PANAMA's $\rho$-function is its use of an SPN structure. Such a KSG design must be motivated by the following simple question: how can a secure cryptographic function be constructed from insecure cryptographic components? For block ciphers (or pseudorandom permutations) there is a *de facto* standard construction, which uses a Feistel network or an SPN as a component (called a round function) and iterates it for mixing. PANAMA is an answer for a KSG. It uses a core mixing function $\rho$ similar to the round function of a block cipher and a large buffer instead of fixed extended keys and iterations of a round function.

On the other hand the function $\lambda$ is a simple linear transformation. The output filter $f$ drops about half of the bits of state $a$ for each round. We call a KSG which satisfies such characteristics PANAMA-like keystream generator (PKSG). This can be formalized in the definition of a PKSG as follows.

**Definition 1** *Consider an internal-state machine consisting of two internal states, namely the state $a$, the buffer $b$, and their update functions $\rho$, $\lambda$. The keystream generator which consists of an internal-state machine $((a, b), (\rho, \lambda))$ and an output filter $f$ is called a* PANAMA***-like keystream generator*** *if it satisfies the following conditions:*

*(1) $\rho$ includes an SPN transformation that uses parts of buffer $b$ as a parameter.*

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

*(2) $\lambda$ is a linear transformation that uses a part of state $a$ as a parameter.*

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

*(3) $f$ outputs a part of state $a$, which is typically no more than $1/2$ of the bits of $a$.*

The first condition characterizes a PKSG, but the other conditions are also necessary. For example, not updating the buffer or outputting all of the state significantly decrease the security [FWT00].

## 2.2 Selection of Components

MUGI is a KSG and has a PKSG structure. In order to select the components for MUGI, we want to build on other strong cryptographic primitives in the literature. As a result we use some components of AES [DR99], which are well evaluated. For example the substitution table S-box and the linear transformation are the same as for AES. Although currently the design of a PKSG is not as straight forward as that of block ciphers, this selection should make MUGI more secure.

## 2.3 The Difference between PANAMA and MUGI

The MUGI design aims to achieve the following two points:

1. Efficiency in hardware implementations. Particularly a gate-efficient implementation must be possible.
2. To make evaluation easier than PANAMA.

To achieve these properties, the basic data size is decreased from 256-bit to 64-bit. And an 8-bit substitution table is adopted to improve the security of $\rho$. In addition, an extended Feistel network is adopted in $\rho$ instead of a simple SPN-structure, in order to simplify the evaluation.

## 3 Specification of MUGI

In this section we give a description of MUGI. MUGI is a KSG with a 128-bit secret key $K$ (a secret parameter) and a 128-bit initial vector $I$ (a public parameter). It generates a 64-bit length random bit string $Out[t]$ for each round.

As we mention in Sect. 2.1 any KSG can be described as the combination of an internal-state machine and an output filter.

First we describe the internal state of MUGI in Sect. 3.2 and the update function in Sect. 3.3. Then we discuss the initialization in Sect. 3.4 and the random number generation in Sect. 3.5.

### 3.1 Input

The basic data size of MUGI is 64 bits, called a *unit* in this paper. MUGI has two inputs as a parameter. One is a 128-bit secret key $K$ and the other one is a 128-bit initial vector $I$. The left and right units of $K$ are denoted by $K_0$ and $K_1$, respectively. $I_0$ and $I_1$ are used in a similar way.

### 3.2 Internal State

MUGI has two internal states, state $a$ and buffer $b$. The state $a$ consists of 3 units denoted by $a_0, a_1, a_2$ from left to right. On the other hand, the buffer $b$ consists of 16 units. Each of them is denoted by $b_0, \ldots, b_{15}$ in the same manner as state $a$.

### 3.3 Update Function

The update function of PKSG consists of $\rho$ and $\lambda$, the update functions of state $a$ and buffer $b$, each of which uses the other internal state as a parameter. In other words the update function $\Upsilon$ of the complete internal state is described as follows:

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

In the following we explain $\rho$ and $\lambda$ of MUGI.

**Core mixing function $\rho$.** $\rho$ is the update function of state $a$. It is a kind of target-heavy Feistel structure [SK96] with two identical F-functions (Fig. 1), it uses buffer $b$ as a parameter. The function $\rho$ can be described as follows:

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

$C_1, C_2$ in the equations above are constants.

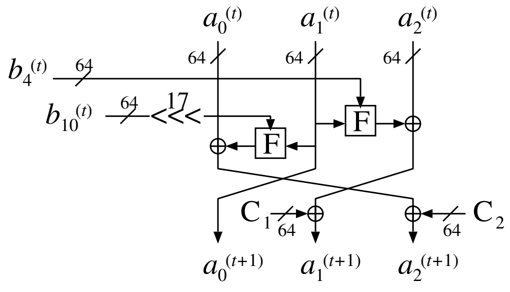

**Fig. 1.** $\rho$-function of MUGI

**F-function.** The F-function consists of a key addition (the data addition from the buffer), a non-linear transformation using the S-box, a linear transformation using the MDS matrix $M$ and a byte shuffling (Fig. 2). The S-box and the MDS matrix are the same as for AES.

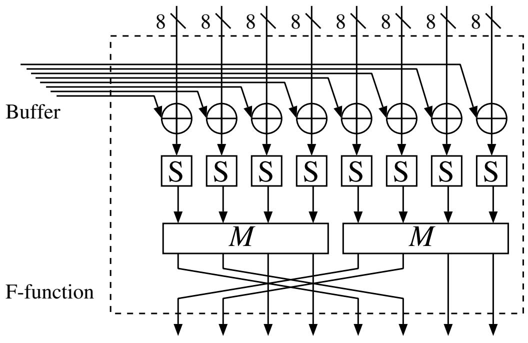

**Fig. 2.** F-function of MUGI

**Buffer update function $\boldsymbol{\lambda}$.** The function $\lambda$ is the update function of buffer $b$, it uses a part of state $a$ as a parameter. $\lambda$ is the linear transformation of $b$ and can be described as follows:

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

## 3.4 Initialization

The initialization of MUGI is divided into three steps. The first step initializes the buffer $b$ with a secret key $K$. The second initializes state $a$ with an initial vector $I$. Finally the whole internal state is mixed.

In the first step the secret key $K$ is extended to 192 bits and it is put into state $a$ as follows:

$$\begin{aligned} a_0^{(t_0)} &= K_0, \\ a_1^{(t_0)} &= K_1, \\ a_2^{(t_0)} &= (K_0 <<< 7) \oplus (K_1 >>> 7) \oplus C_0, \end{aligned}$$

Here time $t_0$ denotes the start of the initialization. The value $C_0$ in the above equation is a constant (see Sect. 3.6). Then follow, a mixing step with only a $\rho$ iteration and the left side unit of each $a^{(t)}$, $a_0^{(t)}$ is put into the buffer $b$ as follows:

$$b_{15-i} = (\rho^{i+1}(a^{(t_0)}, 0))_0$$

In the above equations $\rho^i$ denotes the $i$-th iteration of $\rho$ and $\rho(a, 0)$ means that the data stored into buffer $b$ is not used for this step.

In the second step the mixed state $a(K) := \rho^{16}(a^{(t_0)}, 0)$ and the initial vector $I$ are required. $I$ is added to state $a$ as follows:

$$\begin{aligned} a(K,I)_0 &= a(K)_0 \oplus I_0, \\ a(K,I)_1 &= a(K)_1 \oplus I_1, \\ a(K,I)_2 &= a(K)_2 \oplus (I_0 <<< 7) \oplus (I_1 >>> 7) \oplus C_0, \end{aligned}$$

Then state $a$ is mixed again with 16 rounds of the iteration $\rho$. So the mixed state $a$ can be represented as $\rho^{16}(a(K,I), 0)$.

The last step consists of 16 rounds of the whole update function $\Upsilon$, so $a^{(1)}$, the initialized state with $K, I$, can be written as follows:

$$a^{(1)} = \Upsilon^{16}(\rho^{16}(a(K,I), 0), b(K)),$$

where $b(K)$ denotes the buffer $b$ initialized by the secret key $K$.

## 3.5 Random Number Generation

After the initialization, MUGI generates 64-bit random numbers and transforms the internal state in every iteration. Denote the output of round $t$ as $Out[t]$, then the output is given as follows:

$$Out[t] = a_2^{(t)}$$

In other words MUGI outputs 64 bits of the right side of state $a$ at the beginning of the round process.

The processes from the initialization to the random number generation are summarized in Table 1.

**Table 1.** Schedule of MUGI

| | Round $t$ | Process | Input | Output |
|---|---|---|---|---|
| Initialization | $-49$ | Inputting Key | $K$ | – |
| | $-48, \ldots, -33$ | Mixing (by $\rho$) | – | – |
| | $-32$ | Inputting IV | $I$ | – |
| | $-31, \ldots, -16$ | Mixing (by $\rho$) | – | – |
| | $-15, \ldots, 0$ | Mixing (by $\Upsilon$) | – | – |
| Generating bit strings | $1, \ldots$ | Mixing and Outputting | – | $Out[t]$ |

## 3.6 Constants

The MUGI algorithm uses three constants: $C_0$ in the initialization, and $C_1, C_2$ in $\rho$. They have the following values:

$$\begin{aligned} C_0 &= \texttt{0x6A09E667F3BCC908}, \\ C_1 &= \texttt{0xBB67AE8584CAA73B}, \\ C_2 &= \texttt{0x3C6EF372FE94F82B}. \end{aligned}$$

These are hexadecimal values of $\sqrt{2}$, $\sqrt{3}$, and $\sqrt{5}$ multiplied by $2^{64}$. These constants aim to prevent the invariance of byte-wise equality, and are chosen to ensure that there is no trap-door.

## 4 Security

The security of KSG is reduced to the relationship between input and output bits (or relationship between output bits). All attacks to KSG that improve over exhaustive key search and over exhaustive search over the internal state use some of these relationships and guess the internal state. We consider the possibility that the attacker can observe any kind of relationship, i.e. the condition that the attacker can observe some deviation between input and output bits (or between only output bits) is identified with the success of the attack, even if the attacker cannot get any information about the internal state. This identification comes from the philosophy that the output sequence of the secure KSG should be *unpredictable*. The relationship mentioned above is divided into three cases as follows:

**Randomness.** An attacker fixes a secret key and an initial vector, and then he observes the relation in the output sequence.

**Re-synchronization attack.** An attacker fixes a secret key, and then he observes the relation between initial vectors and output sequences.

**Related-key.** An attacker fixes an initial vector, and then he observes the relation between keys and output sequences. The related-key attack includes observing the relation between keys and initial vectors.

On the other hand exhaustive key searching require $2^{127}$ computations on average to find correct key. So we say an attack is efficient when it costs less than $2^{127}$ encryptions on average to find the correct key.

## 4.1 Randomness of an Output Sequence

The linearity should be one of the most important characteristics in the known evaluation methods. Here 'linearity' does not imply linear complexity, but maximum probability of linear combinations of output bits. We note that searching this linear combination is analogous to search for the best approximation for a block cipher, and apply the evaluation method used in linear cryptanalysis [Ma94]. More specifically, this corresponds to counting active S-boxes in linear approximations for evaluating the linearity of a MUGI output sequence. At the same time applying this technique to PKSGs is more difficult than applying it to block ciphers because the buffer is updated dinamically. Therefore, we give up constructing actual linear approximations and calculate the lower bound of the number of active S-boxes required for any linear approximation.

We denoted the number of active S-boxes of a linear approximation with $\mathcal{AS}$. The maximum linear probability of the MUGI S-box is $2^{-6}$, so it can be assumed that the linear characteristic of the output sequence of MUGI is sufficiently small if there is no linear approximation with $\mathcal{AS} < 22$. Applying this method to MUGI results in the following theorem:

**Theorem 1** *For all linear approximations of MUGI, $\mathcal{AS} \geq 22$.*

Here, we present the proof of this theorem. Constructing a linear approximation which consists of output units can be separated into two steps as follows:

1. Construct a linear approximation of $\rho$.
2. Search a path including the buffer.

We illustrate each step below.

**The linear approximation of $\boldsymbol{\rho}$.** Before starting the evaluation, we give an equivalent transformation of $\rho$ for easier analysis. Figure 3 shows the digest of the transformation. In Fig. 3 the left side F-function is denoted by G; we use this notation just for convinience. First, F can be moved to the left side in the next round. Next, the mask corresponding to an output unit can assume all values, so we separate this part into two masks, an output mask and an input mask. This transformation is not 'equivalent' in the common sense, but it is equivalent in the sense that mask patterns are not changed by the transformation. After that, we remove unnecessary branches. The right side of Fig. 3 shows the transformed $\rho$. Hereafter, "$\rho$" represents the transformed $\rho$. Note that the number of branches drops off to two, and the output masks of the F- and G-functions come directly from the 'input' and 'output' masks, which the attacker can choose.

Figure 4 shows some important paths of $\rho$. Only the five paths shown there assure that the number of active S-boxes is greater than five. The branch number of the matrix $M$ is defined by $\min_{x \neq 0}(w_H(x) + w_H(Mx))$, where $w_H(x)$ denotes the byte-wise Hamming weight of $x$ [Da95]. The branch number of the linear transformation is an important characteristic for the diffusion properties of a block cipher. But for PKSGs, the branch number of the matrix $M$ does

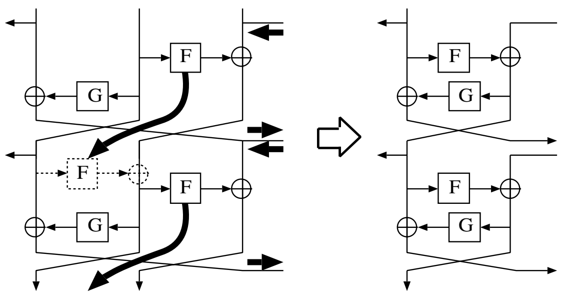

**Fig. 3.** Equivalent transformation of $\rho$

not guarantee a lower bound on the number of active S-boxes for a linear approximation, even if it includes several active F-functions. This property is quite different from that of block ciphers.

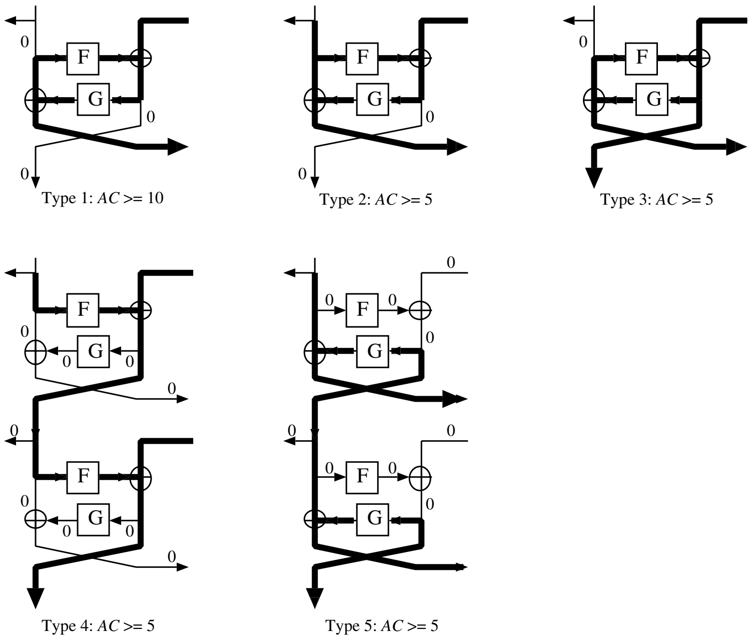

**Fig. 4.** Linear approximation of $\rho$

**Linear path trail of MUGI.** Next, we search for a path including the buffer that gives a linear approximation consisting of only output bits. For PKSGs, the attacker can observe any number of rounds. So it is possible to construct the linear approximation with the outputs of any rounds. Furthermore, some linear approximations may skip intermediate $\rho$-functions, it implies that there is a possibility that to observe more rounds increases the deviation. This feature makes it difficult to search all paths.

Before the starting discussion we define some notation. We denote the first and last round of the path as $t_s$ and $t_e$. The mask that is applied to the data XOR-ed from state $a$ to buffer $b$ is denoted as $\Gamma(D)^{(t)}$. In addition we denote an active F-function as 1, and a zero approximated F-function as 0. For example, when an F-function is active and a G-function is not active in round $t$, we denote this as $\Gamma(a)^{(t)} = (1, 0)$.

First, we pay special attention to the first round and the last round of the path. The value of the input mask for all units of the buffer and their state is zero in the first round, and only the mask for an output unit $Out[t_s]$ is active. Only two paths, Type 1 and 3 in Figure 4, satisfy this condition. The last round is the same as the first round, so the possible paths in round $t_e$ are only those shown as Type 1 and 2.

Next we consider the influence of the buffer to $\rho$. The $\Gamma(D)^{(t)}$ is 0 from round $t_s$ to $t_s + 4$ because all input masks for the first round are 0. In addition, the input mask from the buffer to the G-function must be active, so $\Gamma(D)^{(t_s+5)}$ is active. In a similar manner, $\Gamma(D)^{(t)}$ is 0 at round $t_e - 5 \leq t \leq t_e$ and is active at round $t_e - 6$.

The path search (or the calculation of the lower bound for $\mathcal{AS}$) is divided into several cases according to the mask before or after the round $t_s$ and $t_e$. But we show the proof only for the case that both masks at round $t_s$ and $t_e$ are of Type 1. Other cases can be proved in similar manner. In this case $\mathcal{AS} \geq 20$ because both the first round and the last round are Type 1. In addition, $\Gamma(D)^{(t_e-6)}$ is active.

If there is a round $i$ $(1 \leq i \leq 4)$ such that $(\Gamma(a)^{(t_s+i)}, \Gamma(a)^{(t_s+i+1)}) = ((0,0),(1,1))$, the path includes more active F-functions of Type 1 or 3, so $\mathcal{AS} \geq 25$ is derived. Hence we consider only the case that $\Gamma(a)^{(t_s+i)} = 0$ for all rounds $i$ from 1 to 4. Similarly the mask condition before the last round must be $\Gamma(a)^{(t_e-i)} = 0$ for all rounds $i$ from 1 to 6. Under this condition, $\Gamma(a)^{(t_s+5)} \neq (0,0)$. Additionally, $\Gamma(D)^{(t_s+6)}$ and $\Gamma(D)^{(t_e-6)}$ are active and $\Gamma(D)^{(t_e-7)}$ is equal to 0. So the number of rounds $t_e - t_s$ must be greater than 14. These results and the fact that $\Gamma(D)^{(t_e-6)}$ is active demonstrate that $\Gamma(a)^{(t_e-6)} \neq (0,0)$ or $\Gamma(a)^{(t_e-7)} \neq (0,0)$. Therefore $\mathcal{AS} \geq 22$ is shown in this case.

## 4.2 Re-synchronization Attack and Related-Key Attack

The re-synchronization attack [DGV94] should be the most effective attack against PKSGs, hence we try to apply it to MUGI. Before starting the discussion we give a brief explanation of this type of attack. A re-synchronization attack can be used against keystream generators, which have not only a secret key, but also a public parameter. It is an effective attack if the initialization of the algorithm is too simple. Under the assumption that the secret key is fixed, the attacker first searches for some relationship between the public parameters and the corresponding outputs. If some relationship has a high probability, one can guess information about the secret key from it. For example, linear cryptanalysis on the counter mode of a block cipher is a kind of re-synchronization attack. Estimating the security against related-key attack is the same as re-synchronization, by interchanging initial vectors with secret keys.

We chose differential and linear characteristics, and SQUARE attack [DKR97] variants for evaluating the relationship between inputs and outputs of MUGI. The attacks against block ciphers using these characteristics are well known as differential cryptanalysis [BS93] and linear cryptanalysis [Ma94]. The design of a PKSG, especially its $\rho$ function, is quite similar to a block cipher design. This suggests that the above two statistical properties are well suited for evaluating the relationship between the initial vector $I$ and a corresponding internal state.

**Maximum differential and linear characteristics of iterations of $\rho$.** Now we ignore the XOR to the buffer and output generation, i.e., we consider only the iteration of $\rho$ and evaluate its differential and linear characteristics. We can apply these evaluation methods in the same way as they are applied to block ciphers.

Table 2 shows the minimum number of active F-functions in all units of state $a$ for each attack.

**Table 2.** Number of active F-functions in the differential and linear paths of $\rho$

| Number of rounds | ⋯ | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Differential | ⋯ | 10 | 12 | 12 | 12 | 14 | 16 | 16 | 16 | 18 | 20 | 20 | 20 | 22 |
| Linear | ⋯ | 10 | 12 | 12 | 13 | 14 | 16 | 16 | 17 | 18 | 20 | 20 | 21 | 22 |

*Resistance against a re-synchronization attack*: Table 2 shows the relationship between the initial vector $I$ and corresponding state $a^{(t)}$ transformed by $t$ iterations of $\rho$. It implies that more than 23 iterations of $\rho$ have no differential and linear characteristics with a probability higher than $2^{-128}$.

In the initialization of MUGI, 16 rounds transformed only by $\rho$ are applied after setting the initial vector $I$. Afterwards, 16 rounds transformed by $\Upsilon$ are applied. However, the buffer $b$ influences the differential and linear characteristics of state $a$ only after round $-9$, i.e., 22 rounds after setting $I$. Therefore, we conclude that to observe the deviation due to these characteristics is difficult after round $t > 0$.

On the other hand Table 2 suggests that there are some correlation between initial vector $I$ and some units of corresponding buffer $b$ at round 0. However, the differential characteristic consists of an output sequence and the buffer has more than two buffer-units. The correlation between one of them and $I$ is too small to observe. Therefore, no attacker can exploit that correlation. The conditions for linear cryptanalysis are similar.

*Related-key attacks*: To observe the correlation between keys and the corresponding outputs is more difficult than the correlation between initial vectors and the corresponding outputs because of the first mixing step. So no security flaw can be found by using differential and linear cryptanalysis.

**Square-attack variants.** Because of the highly byte-oriented structure, some of the SQUARE attack [DKR97] variants can be considered. The SQUARE attack is currently the most successful attack against block ciphers with an SPN-structure, e.g., Rijndael, the AES. We examine the applicability of the attack and investigate the possible relations. Consequently we conclude that no variants of the SQUARE attack can reduce the security of MUGI PKSG.

The SQUARE attack against a block cipher is a chosen-plaintext attack where an attacker chooses a number of related plaintext blocks each of which is typically different only in a byte or a word. We call these chosen plaintext set $\Lambda$-set if the word has all values, and we say that the word is *saturated.* Because of the saturation at the input of a non-linear function, the attacker can expect to control the intermediate values to some extent. From the ciphertext side, the attacker partially decrypts the intermediate value which is still controlled because of the saturated plaintext blocks. If the attacker guesses the key for the partial decryption, then the attacker can distinguish the correct and incorrect round keys.

In a stream cipher, an attacker must try to select different value of either key or initial vector values to mount this attack. Therefore the possible applications of the SQUARE attack must be either a related-key cryptanalysis or a chosen initial vector attack.

*Related-key attacks*: At first, we define the model of the attack. We assume that the attacker does not know the key value. To obtain the saturation property, the attacker can *run* a number of key initializations, the keys of which differ only in a part of the key value; in this discussion we will concentrate on key-dependent runs where the keys differ in one word. The attacker cannot observe anything until the pseudorandom number sequence comes out. We check if the attacker may find any properties at the output sequences amongst a number of runs.

The saturated key set will inject the saturation property during the buffer initialization. At first, we investigate how buffers are initialized with the properties. For simplicity, we ignore the key padding rule so that we give the attacker the maximum flexibility for setting the initial state values. Let $\Lambda$ denote the property of an intermediate word such that in each run the concerning word has a different value, i.e., the word is saturated. Let $O$ denote the property that for all runs the value is constant. Also we introduce the weakest property "balanced" denoted by $\Phi$ that means that the XOR-summation over all runs is zero. If the word is neither of them, namely uncontrollable, then we use the notation $*$. If the word triple $(A, B, C)$ has the properties of $\Lambda$, $O$, and $\Phi$ for the word $A$, $B$, $C$, then we write $(A, B, C) \xrightarrow{p} (\Lambda, O, \Phi)$, or $A \xrightarrow{p} \Lambda, B \xrightarrow{p} O$, and $C \xrightarrow{p} \Phi$.

Obviously the most effective word in which to inject the saturation is the word that affects other words the last. We analyze the case of $(a_0, a_1, a_2) \overset{p}{\to} (\Lambda, O, O)$. Remember the output of the $t$-th round is denoted by $(a_0^{(t)}, a_1^{(t)}, a_2^{(t)})$. We simply trace the property and show the results in Table 3.

**Table 3.** The word properties in each intermediate values

| Intermediate value | Word property |
|---|---|
| $(a_0^{(0)}, a_1^{(0)}, a_2^{(0)})$ | $(\Lambda, O, O)$ |
| $(a_0^{(1)}, a_1^{(1)}, a_2^{(1)})$ | $(O, O, \Lambda)$ |
| $(a_0^{(2)}, a_1^{(2)}, a_2^{(2)})$ | $(O, \Lambda, O)$ |
| $(a_0^{(3)}, a_1^{(3)}, a_2^{(3)})$ | $(\Lambda, \Lambda, \Lambda)$ |
| $(a_0^{(4)}, a_1^{(4)}, a_2^{(4)})$ | $(\Lambda, \Phi, \Phi)$ |
| $(a_0^{(5)}, a_1^{(5)}, a_2^{(5)})$ | $(\Phi, *, *)$ |
| $(a_0^{(6+)}, a_1^{(6+)}, a_2^{(6+)})$ | $(*, *, *)$ |

Hence, the initial values of the buffer $b_i$ have the following properties depending on the index $i$:

$$b_i \overset{p}{\to} \begin{cases} O : i = 15, 14, \\ \Lambda : i = 13, 12, \\ \Phi : i = 11, \\ * : i = 10, 9, ..., 0 \end{cases} \tag{1}$$

Note that this does not mean that the attacker is able to control the intermediate value up to $b_{11}$. In fact, $b_{11}$ can be expressed by other buffer values and a single $F$-function evaluation (see the discussion above concerning non-linear buffer relation). However, thanks to the subsequent randomization after initial vector injection, this property must be destroyed before the output sequence is generated. Therefore we believe the related-key attack based on the SQUARE attack does not pose any threat.

*Re-synchronization attacks*: This attack may be more practical than the above related-key cryptanalysis. However, the initial vector does not inject any value to the buffer until the 16-round mixing completes. Taking the number of controllable rounds shown above into account, 16-round mixing is sufficient to destroy the saturation property due to initial vector.

## 5 Implementation

MUGI is designed to be suitable both in software and hardware implementations. In both cases, the implementation achieves a high performance and a low implementation cost. Table 4 and 5 summarize the software and hardware performance respectively. Table 4 shows that the performance in C is a little bit faster than AES.

**Table 4.** Software performance

| Processor | Frequency | OS | Compiler | Performance (cycle/byte) |
|---|---|---|---|---|
| Alpha 21164 | 600MHz | Digital UNIX V4.0B | DEC cc | 9.8 |
| Intel Pentium III | 500MHz | Windows NT 4.0 | Visual C++ 6.0 | 17.7 |

**Table 5.** Hardware performance (Hitachi 0.35 $\mu$m CMOS ASIC library)

| Optimization | Gate size (K gate) | Clock cycle (MHz) | Throughput (Mbps) | Initialization (ns) |
|---|---|---|---|---|
| speed opt. | 26.1 | 45.7 | 2922 | 1095 |
| gate cnt. opt. | 18.0 | 42.3 | 676 | 4590 |
| (3 layers pipelining) | ($\geq$ 19.0) | (126.6) | (2025) | (1531) |

The hardware implementation of MUGI achieves excellent performance, several time faster than AES.

## 6 Conclusion

We have proposed a new keystream generator MUGI built on the idea of PANAMA. MUGI is efficient in both hardware and software. Our security analysis indicates that MUGI is resistant against related-key attacks and resynchronization attacks. But the security of MUGI should be evaluated more. We invite the reader to explore the security of MUGI.

## References

[BS93] E. Biham, A. Shamir, "Differential Cryptanalysis of the Data Encryption Standard," Springer-Verlag, 1993

[CGMPS00] A. Clark, J. Golic, W. Millan, L. Penna, L. Simpson, "The LILI-128 Keystream Generator," *NESSIE project submission*, 2000, available at `http://www.cryptonessie.org`.

[CL01] P. Crowley, S. Lucks, "Bias in the LEVIATHAN Stream Cipher," *Fast Software Encryption, FSE 2001*, Proceedings, pp. 223–230, 2001.

[Da95] J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," Doctoral Dissertation, March 1995, K. U. Leuven.

[DC98] J. Daemen, C. Clapp, "Fast Hashing and Stream Encryption with PANAMA," *Fast Software Encryption, FSE'98*, Springer-Verlag, LNCS 1372, pp.60–74, 1998.

[DGV94] J. Daemen, R. Govaerts, J. Vandewalle, "Resynchronization weaknesses in synchronous stream ciphers," *Advances in Cryptology, Proceedings Eurocrypt'93*, Springer-Verlag, LNCS 765, pp. 159-169, 1994.

[DKR97] J. Daemen, L. Knudsen, V. Rijmen, "The Block Cipher SQUARE," *Fast Software Encryption*, Springer-Verlag, LNCS 1267, pp. 149–165, 1997.

[DR99] J. Daemen, V. Rijmen, "AES Proposal: Rijndael," AES algorithm submission, September 3, 1999, available at `http://www.nist.gov/aes/`.

[Fl01] S. Fluhrer, "Cryptanalysis of the SEAL 3.0 Pseudorandom Function Family," *Fast Software Encryption, FSE 2001*, Proceedings, pp. 142–151, 2001.

[FS01] S. Fluhrer, M. Shamir, "Weaknesses in the Key Scheduling Algorithm of RC4," *Selected in Areas in Cryptography, SAC 2001*, Springer-Verlag, LNCS 2259, pp. 1–24, 2001.

[FWT00] S. Furuya, D. Watanabe, K. Takaragi, "Self-Evaluation Report MULTI-S01," 2000, available at `http://www.sdl.hitachi.co.jp/crypto/s01/index.html`

[JK97] T. Jacobsen and L. R. Knudsen, "The Interpolation Attack on Block Ciphers," *Fast Software Encryption, FSE'97*, Springer-Verlag, LNCS 1267, pp. 28–40, 1997.

[Ku94] L. R. Knudsen, "Truncated and Higher Order Differentials," *Fast Software Encryption, FSE'94*, Springer-Verlag, LNCS 1008, pp. 196–211, 1995.

[Ma94] M. Matsui, "Linear cryptanalysis method for DES cipher," *Advances in Cryptology, Eurocrypt'93*, Springer-Verlag, LNCS 765, pp. 159–169, 1994.

[McF00] D. McGrew, S. Fluhrer, "The stream cipher LEVIATHAN," *NESSIE project submission*, 2000, available at `http://www.cryptonessie.org/`.

[MFI01] M. Mihaljevic, M. Fossorier, H. Imai, "Fast Correlation Attack Algorithm with List Decoding and an Application," *Fast Software Encryption, FSE 2001*, Proceedings, pp. 208–222, 2001.

[RC94] P. Rogaway, D. Coppersmith, "A Software-Optimized Encryption Algorithm," *Fast Software Encryption, FSE'94*, Springer-Verlag, LNCS 809, pp. 56–63, 1994.

[RC98] P. Rogaway, D. Coppersmith, "A Software-Optimized Encryption Algorithm," *Journal fo Cryptography*, Vol. 11, No. 4, pp. 273–287, 1998.

[RRPV01] V. Rijmen, B. Van Rompay, B. Preneel, J. Vandewalle, "Producing Collisions for PANAMA," *Fast Software Encryption, FSE 2001*, proceedings, pp. 39–53, 2001.

[Sc96] B. Schneier, *Applied Cryptography*, Second Edition, John Wiley & Sons, pp. 397-398, 1996.

[SK96] B. Schneier, J. Kelsey, "Unbalanced Feistel Networks and Block Cipher Design," *Fast Software Encryption, FSE'96*, Springer-Verlag, LNCS 1039, pp. 121–144, 1996.

[Spec] D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, *MUGI Pseudorandom number generator, Specification*, 2001, available at
`http://www.sdl.hitachi.co.jp/crypto/mugi/index-e.html`.

[Eval] D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, *MUGI Pseudorandom number generator, Self Evaluation*, 2001, available at
`http://www.sdl.hitachi.co.jp/crypto/mugi/index-e.html`.

## A Test Vector

```
key[16] =
{0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f}
iv[16]  =
{0xf0 0xe0 0xd0 0xc0 0xb0 0xa0 0x90 0x80 0x70 0x60 0x50 0x40 0x30 0x20 0x10 0x00}

after key input:
state  a = 0001020304050607  08090a0b0c0d0e0f  7498f5f1e727d094
buffer b =
0000000000000000  0000000000000000  0000000000000000  0000000000000000
0000000000000000  0000000000000000  0000000000000000  0000000000000000
0000000000000000  0000000000000000  0000000000000000  0000000000000000
```

```
0000000000000000  0000000000000000  0000000000000000  0000000000000000

after the first 16 rounds mixing:
state  a = 7dea261cb61d4fea  eafb528479bb687d  eb8189612089ff0b
buffer b =
7dea261cb61d4fea  bfe2485ac2696cc7  c905d08f50fa71db  fd5755df9cc0ceb9
5cc4835080bc5321  dfbbb88c02c9c80a  591a6857e3112cee  20ead0479e63cdc3
2d13c00221057d8d  b36b4d944f5d04cb  738177859f3210f6  c08ee4dcb2d08591
9c0c2097edb20067  09671cfbcfaa95fb  9724d9144c5d8926  08090a0b0c0d0e0f

after iv input:
state  a = 8d0af6dc06bddf6a  9a9b02c4499b787d  f100cffe031d365b
buffer b =
7dea261cb61d4fea  bfe2485ac2696cc7  c905d08f50fa71db  fd5755df9cc0ceb9
5cc4835080bc5321  dfbbb88c02c9c80a  591a6857e3112cee  20ead0479e63cdc3
2d13c00221057d8d  b36b4d944f5d04cb  738177859f3210f6  c08ee4dcb2d08591
9c0c2097edb20067  09671cfbcfaa95fb  9724d9144c5d8926  08090a0b0c0d0e0f

after the second 16 rounds mixing:
state  a = 4e466dffcb92db48  f5eb67b928359d8b  5d3c31a0af9cd78f
buffer b =
7dea261cb61d4fea  bfe2485ac2696cc7  c905d08f50fa71db  fd5755df9cc0ceb9
5cc4835080bc5321  dfbbb88c02c9c80a  591a6857e3112cee  20ead0479e63cdc3
2d13c00221057d8d  b36b4d944f5d04cb  738177859f3210f6  c08ee4dcb2d08591
9c0c2097edb20067  09671cfbcfaa95fb  9724d9144c5d8926  08090a0b0c0d0e0f

after the whole initialization:
state  a = 0ce5a4d1a0cbc0f7  316993816117e50f  bc62430614b79b71
buffer b =
d25c6643a9dabd67  e893c5b5a5b2ff2b  ce840df556562dc6  4210def4ccf1b145
5eda7c5b0dbf1554  d3e8a809b214218a  d42bcb0bb4811480  76d9c281df20192d
3dc6c6bc876beb72  39d84df58f8840e2  cd7fe2794367de6c  680920245819a4f5
f5e9e609dd8e3cc3  9cf94157cf512603  871323e1d70caa2b  0b6bb4c0466c7aba

output =
bc62430614b79b71  71a66681c35542de  7aba5b4fb80e82d7  0b96982890b6e143
4930b5d033157f46  b96ed8499a282645  dbeb1ef16d329b15  34a9192c4ddcf34e
...
```