MUGI Pseudorandom Number Generator

Self-evaluation Report Ver. 1.1

Hitachi, Ltd.
2001. 12. 18

Contents

1 Introduction

This documentation gives the royal summary of designers' assessments on MUGI pseudorandom number generator. MUGI is a pseudorandom number generator, especially for the purpose of the stream cipher. As is often the case with other cryptographic primitives, the security of a pseudorandom number generator is considered high only if the underground evaluation is good. In this documentation the designers of the MUGI contribute as many information used to design as possible that would help future assessment. The documentation is aimed to give the evidence of its expected security and performance.

This document is organized as follows: In Section 2 we give some notations and terms. The specification is given in [Spec]. In Section 3 we present some results about the security of MUGI. In Section 4 we show the implementation of MUGI on software and hardware.

As the result, we conclude MUGI can be one of reliable and efficient cryptographic primitives applicable to encryption and authentication functions.

2 Preliminaries

In this section we describe notations and abbreviations used in this document.

2.1 Notations

$\oplus$	bit-wise XOR
$\wedge$	bit-wise AND
$\|\|$	concatenation of two strings
$>>> n$	n-bit right rotation in a 64-bit register
$<<< n$	n-bit left rotation in a 64-bit register
`0x`	prefix indicating hexadecimal representation

2.2 Abbreviations

LFSR	Linear Feedback Shift Register
PRNG	Pseudorandom Number Generator
PKSG	Panama-like Keystream Generator (See [Spec].)

3 Security

In this section we describe the summary of the security assessment of MUGI key-stream generator, which is a kind of PKSG. The prototype Panama was designed by J. Daemen and C. Clapp, and is a cryptographic module for a hash function and a stream cipher [DC98]. PKSG is a generalization of a stream-cipher mode of Panama. The detail definition of PKSG is given in [Spec].

We think of PKSG as the combination of a LFSR and a round function. The considered LFSR is relatively simpler and much larger than ones used in the currently known LFSR-based key stream generators, meanwhile its round function is similar to ones used in a block cipher. Because of this structure there are no reliable and thorough methods to evaluate the security of PKSG at present. In this section we examine the security of MUGI from many aspects.

Generally any PRNGs must satisfy following two requirements:

(1) The output sequence has good enough randomness.

(2) Any two output sequences generated by different initial data are significantly different.

For the assessment of (1), namely the randomness of the output sequence, we take two kinds of approaches. The first is based on some numerical tests of the actual output sequence of MUGI. The results are shown in 3.1. The second approach has been done with some traditional cryptanalysis against MUGI, which is discussed in 3.2. Subsequently we describe the new method that is suitable for evaluating the randomness of PKSGs. The detail of this method and the result are given in 3.3.

For the evaluations as for (2), we take two inputs into account, namely, the secret key and the initial vector. We treat this separately in two cases. In first case, the secret key is fixed. We apply various initial vectors and observe the randomization of their output sequences. The general study about this kind of attack can be found in [DGV94]. In this documentation we leave results in 3.4.

The second case is the one where the variable secret keys are analysed under a fixed initial vector. This kind of attack can be considered as the related-key attack. We describe the results in **??**. Furthermore we note about some simple relationship between a secret key and a initial vector in the same part.

3.1 Randomness test based on FIPS 140-1

In this section we present the results of our statistical randomness tests. The tests we used to examine the randomness of the output sequences of MUGI can be found in FIPS 140-1 [FIPS]. The tests are

(1) Mono-bit frequency test

(2) Poker test (four-bit frequency test)

(3) Run test(including detection of a long run)

They were designed for examining short sequences so they are not suited to sequences used for stream ciphers. We used frequency test using longer sequences than what is specified with the tests FIPS.

3.1.1 Frequency test

Here we checked the 1-, 2-,4-, and 8-bit frequencies. The method we used was described by Knuth [Kn81]. Additionally, we applied these tests to 2^{22}-,2^{26}-, and 2^{30}-bit-length sequences to observe the effect of the sequences length. We generated 512 sets of initial data by Rijndael. In detail we used output of random number generator in the standard C library as the input of Rijndael and used output of them as the initial data.
Table 1,Table 2,Table 3, and Table 4 shows the results of the frequency tests.

The values in the tables represent how many of the initial data were regarded as wrong in each test and the rejection probabilities. For instance, in the mono-bit frequency test to 2^{30}-bit-length sequences, 5 of 512 output sequences generated from the initial data were distinguished from truly random sequences with probability 99%.

Table 1: Results of randomness test (1-bit frequency)

	number of rejected key (/512)	
data length(bit)	Rate of rejection 0.05	Rate of rejection 0.01
2^{22}	25	6
2^{26}	20	4
2^{30}	22	5

Table 2: Results of randomness test (2-bit frequency)

	number of rejected key (/512)	
data length(bit)	Rate of rejection 0.05	Rate of rejection 0.01
2^{22}	19	3
2^{26}	23	8
2^{30}	18	3

Table 3: Results of randomness test (4-bit frequency)

	number of rejected key (/512)	
data length(bit)	Rate of rejection 0.05	Rate of rejection 0.01
2^{22}	18	6
2^{26}	31	6
2^{30}	16	3

The ratio of the number of rejected initial data to the number of tested ones nearly equalled to the rate of rejection in each test.

Table 4: Results of randomness test (8-bit frequency)

	number of rejected key (/512)	
data length(bit)	Rate of rejection 0.05	Rate of rejection 0.01
2^{22}	17	6
2^{26}	18	6
2^{30}	18	1

Table 5: Results of randomness test (Run test)

	number of rejected key(/512)	
data length(bit)	Rate of rejection 0.05	Rate of rejection 0.01
2^{22}	29	3

3.1.2 Run test

The method we used was described by [MOV97]. The method we generated initial data by was same as 3.1.1.
The values in the tables represent how many of the initial data were regarded as wrong in each test and the rejection probabilities. For instance, in the test to 2^{22}-bit-length sequences, 3 of 512 output sequences generated from the initial data were distinguished from truly random sequences with probability 99%.

Table 5 shows that the ratio of the number of rejected initial data streams to the number of tested ones nearly equalled the rate of rejection in the test. On equal terms with above tests the longest run was 31 long and the number of the run was 1. The expected value that a run of length 31 exists in a 2^{22}-bit stream is 2^{-11}. Detection of a run of length 31 is expected once by 4 times because we generated 512 the set of initial data. Hence this result is appropriate.

3.2 Comments on known evaluation methods

3.2.1 Period

A secure PRNG requires a long period matching the length of the secret parameter. For each secret key K and each initial vector I, the MUGI output sequence should have a period longer than 2^{128} because the MUGI key length is 128 bits.

Estimating the period of the MUGI output sequences is difficult because its update function is non-linear. However, its huge internal state and the design of ρ imply that the period of its output sequence is longer than that generated by the OFB-mode of a 128-bit block cipher.

3.2.2 Linear complexity

The linear complexity is efficient when the update function is LFSR. Therefore, we think applying the Berlekamp-Massey algorithm to MUGI is difficult.

3.2.3 Divide-and-conquer attack

A divide-and-conquer attack can be used when the PRNG has several internal states and one of them has an independent update function. Using this assumption, an attacker hipothesizes this part of the internal state at first. Then he can work out this part in all round.

However, a PKSG generally has a huge internal state and it is impossible to isolate its update function. Thus, we conclude that applying divide-and-conquer attack to MUGI is difficult.

The only possibility using this kind of attack against MUGI is by using a re-synchronization method. If the initialization is insufficient, some of the internal states may be conjectured with significant probability. This kind of attack is considered in 3.4.3.

3.3 Optimal evaluation for PKSG

There are various techniques for evaluating the randomness of a bit string, but almost all of them require actual bit strings and calculate their statistical property. When the bit strings are generated by a PRNG based on LFSR,

many theoretical attacks, such as a correlation attack, are possible. However, as we mentioned in 3.2, these techniques can not be used against the PKSGs output sequence.

We note that the PKSG's structure is similar to that of block ciphers (See [WFT01] [WFST01a].) and examine the possibility of using linear cryptanalysis, which is known as an effective attack against all block ciphers.

3.3.1 Differential / linear characteristics of F-function

The MUGI F-function has a SPN-structure and consists of key-XORing, byte-wise non-linear transformation using S-box, byte-wise linear transformation (described as a 4×4 matrix M), and bytes shuffling (see Figure 1).

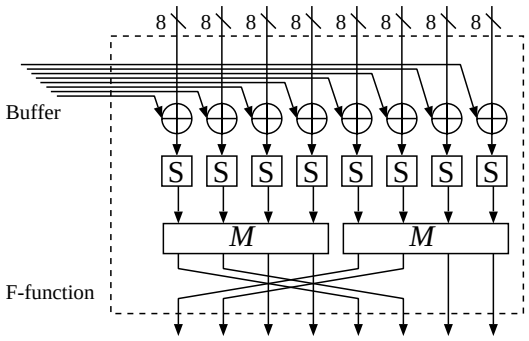

Figure 1: F-function of MUGI

The S-box and the matrix M are the same as those used in AES [DR99]. Hence, the maximum differential and linear probability of S-box are 2^{-6} and the branch number of the matrix M is five. Note that the linear probability is normalized.

Next consider a differential characteristic having two F-functions as in the following:

$$\Delta_0 \xrightarrow{\mathrm{F}} \Delta_1 \xrightarrow{\mathrm{F}} \Delta_2$$

The probability of this characteristic is less than 2^{-30}. Furthermore when $\Delta_0 = \Delta_2$, more than ten active S-boxes are required in this differential characteristic. Therefore the probability is less than 2^{-60}.

The maximum probability of a linear characteristic that has two F-functions is the same as that for the differential characteristic.

$$\Gamma_0 \xrightarrow{\mathrm{F}} \Gamma_1 \xrightarrow{\mathrm{F}} \Gamma_2$$

The probability of the linear characteristic above is less than 2^{-30}. If $\Gamma_0 = \Gamma_2$, the probability is less than 2^{-60}.

3.3.2 Application of linear cryptanalysis

Linear cryptanalysis is an attack used against block ciphers. It observes a linear combination of plural bits that consists of the input block and the output block. The attack is effective when there is a linear combination with a value that is not exactly uniform. This technique offers a means to observe the linear correlation between the input and the output blocks.

The evaluation technique that we propose calculates a linear combination of plural bits that consists of some output units. This requires constructing a linear approximation that consists of output units. Applying this technique to PKSGs is more difficult than applying to block ciphers because the buffer (This corresponds to the round keys for block ciphers) is dynamically updated. Therefore, we give up constructing actual linear approximations and examine calculating the lower bound of the number of active S-boxes required for any possible linear approximation.

Consequently, this evaluation method cannot distinguish the output sequence from a truly random sequence.

Theorem 1 *When the lower bound of the number of active S-box for MUGI is more than 22.*

Here, we present the proof of this theorem. As previously mentioned, we have to construct a linear approximation consisting of only output units for using linear cryptanalysis against PKSG. This approach can be separated into two steps as follows:

1. Constructing a linear approximation of ρ
2. Searching a path including the buffer

We illustrate each step below.

The linear approximation of ρ

First, we transform ρ for easy evaluation as shown in Figure 2. Hereafter, "ρ" represents transformed ρ.

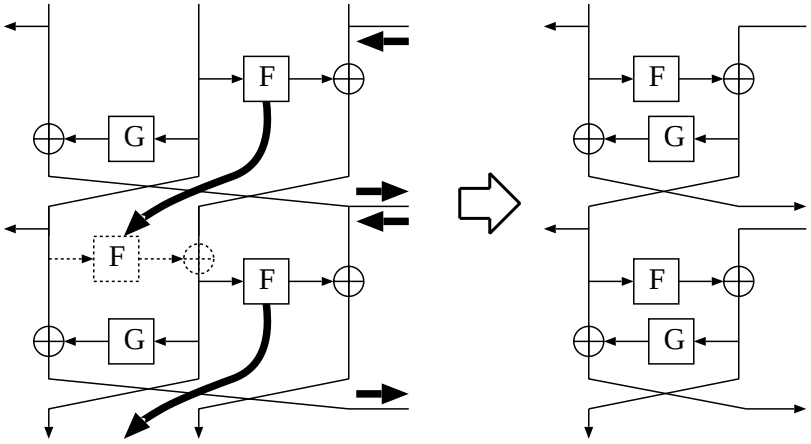

Figure 2: Equivalent transformation of ρ

Any linear approximations consisting of one or plural rounds are given as the combination of the two following approximations (see Figure 9 for reference):

$$\Gamma_1 \cdot a_0^{(t)} \oplus \Gamma_2 \cdot a_2^{(t)} = \Gamma_2 \cdot a_0^{(t+1)}, \quad (1)$$

$$\Gamma_1 \cdot a_0^{(t)} \oplus \Gamma_1 \cdot a_2^{(t+1)} = \Gamma_2 \cdot a_0^{(t+1)} \quad (2)$$

The symbol "$\cdot$" indicates the inner product. The thick line in Figure 9 illustrates the active path. Each of the equation (1), (2) gives a linear approximation consisting of only one F- (or G-) function.

We show some important paths in Appendix (Figure 10). Only the five paths shown there assure that the number of active S-boxes is greater than five. Note that the branch number of matrix M does not assure the number of active S-boxes for a linear approximation, even if it includes several active F-functions. This property is quite different from those of block ciphers.

Next, we search for a path including the buffer that gives a linear approximation consisting of only output units. For PKSGs, the number of rounds

is not given, i.e. they have the possibility of constructing a linear approximation consisting of any rounds. This feature makes it difficult to search all paths.

Conditions required of all linear approximations

We pay special attention to two rounds, the first and last round of the path. These rounds and their neighbors must meet some required conditions. For example, all input masks of internal states must be 0. We classify the paths by using these conditions and calculate the lower bound of active S-boxes for each case. Hereafter, we denote the first round as t_s, the last round as t_e, and the lower bound of active S-boxes of each path as $\mathcal{AS}$. The maximum linear probability of the MUGI S-box is 2^{-6}, so it can be assumed that the linear correlation of the output sequence of MUGI is small enough if there is no linear approximation with $\mathcal{AS} < 22$.

In addition, noticing the active mask of the data from state a to buffer b, we easily characterize the iteration expression.

the mask that is applied to the data XOR-ed from state a to buffer b must concatenate plural paths. We denote this mask as $\Gamma(D)^{(t)}$.

First, we consider the first round and last round of the path. The value of the input mask for all units of the buffer and their state is zero at 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 10, satisfy this condition. The last round is the same as the first round, so the possible paths at round t_e are only those shown as Type 1 and 2.

Next we consider the influence of the buffer to ρ. The $\Gamma(D)^{(t)}$ is 0 from round t_s to t_s+4 because all input masks at the first round are 0. In addition, the input mask from the buffer to 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 $t_e - 6$.

Calculating the lower bound of the number of active S-boxes

Hereafter 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 at round t, we denote this as $\Gamma(a)^{(t)} = (1, 0)$

The search path (or calculating the lower bound of $\mathcal{AS}$) is separated in several cases. The first condition are the followings:

Condition T1: There is a round $i(2 \leq i \leq 7)$ such that

$$(\Gamma(a)^{(t_e-i)}, \Gamma(a)^{(t_e-i+1)}) = ((0,0),(1,1)).$$

Condition T2: For all rounds $i(1 \leq i \leq 7)$,

$$\Gamma(a)^{(t_e-i)} \neq (0,0).$$

Condition T1 and T2 are complementary.

We consider a similar condition at rounds t_s to $t_s + 4$ when

$$(\Gamma(a)^{(t_s)}, \Gamma(a)^{(t_s+1)}) = ((1,1),(0,0))$$

is satisfied:

Condition H1: $\Gamma(a)^{(t_s+i)} = (0,0), \quad 1 \leq i \leq 4,$

Condition H2: Complemented condition of H1.

Now we calculate the lower bound of $\mathcal{AS}$ in the following four cases:

A. First round: Type 1, Last round: Type 1

B. First round: Type 1, Last round: Type 2

C. First round: Type 3, Last round: Type 1

D. First round: Type 3, Last round: Type 2.

(1) Case A

In this case $\mathcal{AS} \geq 20$ because both the first round and last round are Type 1. In addition, $\Gamma(D)^{(t_e-11)}$ is active.

If the condition H2 is satisfied, the path includes more active F-functions with Type 1 or 3, so $\mathcal{AS} \geq 25$ is derived.

On the other hand, if the path satisfies condition H1, $\Gamma(a)^{t_s+5} \neq (0,0)$. Additionally, $\Gamma(D)^{(t_s+6)}$ and $\Gamma(D)^{(t_e-6)}$ are active and $\Gamma(D)^{(t_e-7)}$ is a zero mask. 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 $\mathcal{AS} \geq 22$ in this case.

(2) Case B

In this case no less than 15 active S-boxes are assured.

First, we consider the case when the condition T1 is satisfied. Then $\mathcal{AS} \geq 20$ because the path includes another path of Type 3. If $\Gamma(a)^{(t_e-6)} \neq (0,0)$, obviously $\mathcal{AS}$ is not less than 25. Additionally, if the round of Type 2 (It must occur because of the condition T1.) is located before round $t_e - 4$, $\mathcal{AS} \geq 22$. Therefore, we only have to consider the possibility of $\Gamma(a)^{(t_e-2)} = 0$. Then $\Gamma(D)^{(t_e-10)}$ is active, so $\mathcal{AS} \geq 22$ is demonstrated in the same manner as in the last half of case A.

Next we consider the assumption of T2, $\mathcal{AS} \geq 21$ by definition. In this case, if the path includes Type 4 or 5, $\mathcal{AS} \geq 24$ is derived. Else, one $\Gamma(D)^{(t)}$ is active every 2 rounds at round $t_e - 12$ to $t_e - 7$. Thus, $\mathcal{AS}$ is not less than 24.

(3) Case C

Here, four more active S-boxes are added to 15 active S-boxes derived from the conditions for case C. At the rounds between $t_s + 1$ and $t_s + 4$, all rounds have an active F-function or there is at least one Type 3 path.

Furthermore, in a similar manner to that shown above, the path around the last round can be restricted as follows:

$$\Gamma(a)^{(t_e-i)} = (0,0), \quad 1 \leq i \leq 6.$$

Then

$$\Gamma(D)^{(t_e-i)} = 0, \quad 7 \leq i \leq 9.$$

So, the rounds between $t_e - 10$ and $t_e - 7$ must have not less than four active S-boxes. This equals $\mathcal{AS} \geq 23$.

(4) Case D

In this case no less than 19 active S-boxes are assured. If the condition T2 is satisfied $\mathcal{AS} \geq 22$ in the same manner as in Case C.

Suppose condition T1 is satisfied and the path of Type 3 occurs at round $t_e - i_0$. The path between round $t_e - i_0 + 1$ and $t_e - 1$ assures

no less than $i_0 - 1$ active S-boxes, $\mathcal{AS} \geq 19 + (i_0 - 1)$. Therefore, we can require that $i_0 \leq 3$ in the same manner as in Case B.

Assuming that $\Gamma(D)^{(t_e-10)}$ is active, i.e. the round $t_s + 5$ is not equal to $t_e - 7$. We have not counted the active S-box in round $t_e - 7$. Thus, we can once again require that $i_0 \leq 2$. Iterating in a similar manner derives that $\mathcal{AS} \leq 22$ in all cases.

3.3.3 Other attacks

Now we discuss some other attacks against block ciphers such as differential cryptanalysis, higher order differential attack, and interpolation attack. All of these attacks are chosen plaintext attacks. Applying these attacks to PKSG should be difficult (We consider them here as a method of evaluating its randomness).

Generally, it is impossible to get the required chosen plaintexts from the output sequence. For example, if the attacker can construct a distinguisher that consists of 16 output rounds, the number of the possible outputs is $2^{64\times16}$. This means that the computational complexity required to get one chosen plaintext is more than $2^{64\times8}$.

In addition, consider using differential cryptanalysis as an example. When the attacker searches the differential characteristics as described in 3.3.2, the attacker assumes the possibility of being able to observe all of the internal states at some round. This assumption is not valid if the initialization is sufficient.

3.4 Re-synchronization attack on MUGI

Next we consider the possibility of using re-synchronization attack [DGV94] against MUGI.

First, we need to make a brief explanation of re-synchronization attack. 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 insufficient. Under the assumption that the secret key is fixed, the attacker first searches for some relationship between the public parameters and corresponding outputs. If some relationship has a high probability, he can guess information about the

secret key from it. For example, linear cryptanalysis on the counter mode of a block cipher is a type of re-synchronization attack.

We chose differential and linear characteristics for evaluating the relationship between inputs and outputs. 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 ρ function, is quite similar to a block ciphers' design. This suggests that these two statistical properties are well suited for evaluating the relationship between the initial vector I and a corresponding internal state.

3.4.1 Differential and linear path of ρ without outputs

In our discussion, we neglect XOR to the buffer and outputting, i.e., we consider only the iteration of ρ and evaluate its differential and linear characteristics. We can apply these evaluation methods in the same way as they are applied to block ciphers.

In our evaluation, we search all differential and linear paths by unit and check on the active F-functions. The maximum differential and linear characteristics are 2^{-6} and the branch number of the linear transform is five. The number of active F-functions does not assure the number of active S-boxes because the F-function consists of a one-layer SPN structure. However, if the number of active F-functions is not less than ten for each unit $a_i^{(t)}$, it can be considered the relation between the initial vector I and the state $a^{(t)}$ at round t is small enough.

Table 6: Number of active F-functions in the differential and linear characteristics of ρ

Number of rounds	$\cdots$	11	12	13	14	15	16	17	18	19	20	21
Differential	$\cdots$	4	5	6	6	6	7	8	8	8	9	10
Linear	$\cdots$	4	5	6	6	6	7	8	8	8	9	10

Table 6 shows the minimum number of active F-functions in all units of state a for each attack. Note that we allow for all condition of the initial

data of state a in the calculations above.

3.4.2 Resistance against re-synchronization attack

Table 6 shows the relationship between the initial vector I and corresponding state $a^{(t)}$ transformed by t iterations of ρ. It implies that more than 21 iterations of ρ have no differential and linear characteristics with a probability higher than 2^{-128}.

In the initialization of MUGI, 16 rounds transformed only by ρ are applied after setting the initial vector I. Afterwards, 16 rounds transformed by *Update* are applied. However, 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 the relationship is too small to observe after round $t > 0$.

3.4.3 Combination of divide-and-conquer attack and re-synchronization attack

Table 6 suggests that some units of buffer b at round 0 have a little relation to corresponding initial vector I. However, the differential characteristic consists of an output sequence and the buffer has more than two buffer-units. The relationship between any of them and I is quite small. Therefore, no attacker can observe that relationship. The conditions for linear cryptanalysis are the same.

3.5 Design and analysis of the key setup

We designed the key setup algorithm so that the resultant initialization of the buffer and the state is enough randomized to generate a secure pseudorandom sequence after the initialization.

We treat the evaluation of the key setup, dividing four aspects, each of which is discussed independently.

3.5.1 (Im)possibility of the linearly-keyed buffer

Generally speaking, the buffer value is initialized non-linearly with respect to the raw key value because of the update function ρ. However in some special

cases, the non-linearity may reduce. One of those cases are ones when many inputs to the ρ functions are the same. This may cause the extremely simple linear relations between the input (the raw key) and the output (the initialized buffer values). In this part of the document, we consider the possibility of keys that initializes buffer with extremely simple linear relations.

There are two stages in the buffer initialization, namely (A) the key-dependent initialization, and (B) the randomization after setting the initial vector, IV hereafter. At first, we note the remark, if the algorithm takes the key such that the buffer initialization by (A) has sufficiently high non-linearity, then it must hardly happen that the resultant buffer (just before outputting the pseudorandom sequence) contains linear relations (with no more than 16 word unknown variables). ¿From this remark, the analysis here has two objectives: verification of the fact that the used function is sufficiently non-linear for most of the key space, and the size of the keys that generate the linear buffer is small enough and negligible.

We also remark that for those limited number of *weak* keys (in the sense of the linear buffering) there are another non-linear randomization onto all buffer value after injection of the IV. This contributes further mixing of buffer and state properties. Consequently we conclude that the key setup sufficiently randomizes the buffer with respect to the linear-buffer initialization.

Case 1:(One-round iteration)

Theorem 2 *ρ function doesn't have any fixed points, i.e., no key generates the same output for all buffer units.*

Proof
Let (a_0, a_1, a_2) and (b_0, b_1, b_2) be the input and the output of the ρ function for the key setup. Remember that in this stage the buffer injection to the F function is zero. The definition of ρ is followed by those relations between a and b:

$$b_0 = a_1, \tag{3}$$
$$b_1 = a_2 \oplus F(a_1, 0) \oplus C_1, \tag{4}$$
$$b_2 = a_0 \oplus F(a_1, 0) \oplus C_2, \tag{5}$$

If (a_0, a_1, a_2) is the fix point, then the following relation must hold as well.

$$b_i = a_i, \qquad i = 0, 1, 2. \tag{6}$$

¿From Equations (3), (4), (5), and (6), we have the following condition for the fix point a.

$$C_1 = C_2.$$

¿From the definition of C_1 and C_2, the above condition is always false. Hence, the ρ function doesn't have any fixed points.

Case 2: (Two-round iteration)

Theorem 3 *A structure with two iterations of the ρ function doesn't have many fixed points, i.e., no obvious weak key classes with respect to the two-value initialized buffer do not exist.*

Proof
The two-round structure of the ρ function itself contains a number of the fixed points. However, the padding rule excludes most of them from the possible input.

In this proof of the theorem, we take the padding rule into account. Let a_0 and a_1 to be the upper and lower half of the secret key. Then the padded key word a_2 is defined as follows:

$$(a_0 <<< 7) \oplus (a_1 >>> 7) \oplus C_0.$$

Noticing the linearity and independence of the a_2 generation. When we study the properties of the iterative keys, we can replace the original key embedding with the following one without loss of generality:

$$\begin{aligned} a_0 &= ((a_1 >>> 7) \oplus C_0 \oplus a_2) >>> 7, \\ a_1 &= [\text{the upper half of the secret key}], \\ a_2 &= [\text{the lower half of the secret key}]. \end{aligned}$$

Let $\alpha = F(a_1, 0)$, and $\beta = F(a_2 \oplus \alpha \oplus C_1, 0)$. The output of the two-round ρ function is deterministically described as

$$(a_2 \oplus \alpha \oplus C_1, a_0 \oplus \alpha \oplus C_2 \oplus \beta \oplus C_1, a_1 \oplus \beta \oplus C_2).$$

Therefore, the necessary conditions for the triple (a_0, a_1, a_2) to generate the two-round iterative output is

$$\begin{aligned} a_1 &= \text{arbitrarily chosen}, \\ a_2 &= F^{-1}(\beta, 0) \oplus C_1 \oplus F(a_1, 0), \\ a_0 &= a_1 \oplus C_1 \oplus \beta \oplus C_2 \oplus \alpha \\ &= a_2 \oplus C_1 \oplus \alpha. \end{aligned}$$

For a fixed a_1 value (2^{64} possibilities), a_2 has one possibility in average because a_2 is defined by the equation

$$a_2 = f^{-1}(a_2 \oplus C_a, 0) \oplus C_b,$$

where C_a and C_b are a_1 dependent constant values. a_0 is uniquely determined by a_1 and a_2. In total, without the consideration of the padding rule, the triple (a_0, a_1, a_2) has about 2^{64} two-round iterative buffer initialization. Our estimation is quite rough but the most of those triples are impossible to generate out of the secret key due to the padding rule. We expect the size to generate the class of the weak key is too small to mount a meaningful attack.

Case 3: (Three-round iteration)

Theorem 4 *A structure with three iterations of the ρ function does not have any fixed points.*

Proof
In this part, the proof does not make use of the padding rule. Let the triple (a_0, a_1, a_2) be the initial secret key. Then, the resultant state value after three rounds is $(a_0^{(3)}, a_1^{(3)}, a_2^{(3)})$, where

$$\begin{aligned} a_0^{(3)} &= a_0 \oplus \alpha \oplus \beta \oplus C_1 \oplus C_2, \\ a_1^{(3)} &= a_1 \oplus \beta \oplus \gamma \oplus C_1 \oplus C_2, \\ a_2^{(3)} &= a_2 \oplus \gamma \oplus \alpha \oplus C_1 \oplus C_2, \\ \alpha &= F(a_1, 0), \\ \beta &= F(a_2 \oplus \alpha \oplus C_1, 0), \\ \gamma &= F(a_0 \oplus \alpha \oplus \beta \oplus C_1 \oplus C_2, 0). \end{aligned}$$

If the triple (a_0, a_1, a_2) generates three-round iterative output, then

$$(a_0, a_1, a_2) = (a_0^{(3)}, a_1^{(3)}, a_1^{(3)}).$$

Therefore we have those three necessary and sufficient conditions:

$$\alpha \oplus \beta = C_1 \oplus C_2, \quad (7)$$
$$\beta \oplus \gamma = C_1 \oplus C_2, \quad (8)$$
$$\gamma \oplus \alpha = C_1 \oplus C_2. \quad (9)$$

¿From Equations (7), (8), we have

$$\gamma \oplus \alpha = 0. \quad (10)$$

This equation (Equation (10)) conflicts the rest of the condition specified by Equation (9). Therefore no key generates the three-round iterative output.

Case 4: (Considerations on more than 3-round iteration) We also considered the cases more than three. However, because of the following reasons we conclude that any case does not contain any weak key class for which an attacker can detect more efficient than the weak-key class exhaustive search.

The size of the applicable secret keys: Normally the necessary condition to generate the iterative output specifies two complicated word-wise equations, that means only 2^{64} possible initial 192-bit states may expose this property in average. As is the case with **Case 2**, the padding rule should effectively avoid the weak-key classes of such keys.

The effectiveness of the iteratively initialized buffer: The effectiveness of the iteratively initialized buffer must be reduced due to the subsequent buffer-state mixing. Moreover, the buffer contains at least four values that looks effectively avoid the simple properties during the initialization.

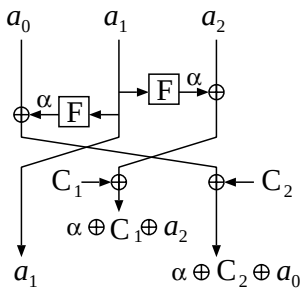

Figure 3: One-round iteration

3.5.2 Square-attack variants

Because of the highly byte-oriented structure, some of the SQUARE attack [DKR97] variants can be considered. The SQUARE attack is the one that is currently most successful attack against the block ciphers with the SPN-structure, e.g., Rijndael, the proposed AES. We examine the applicability of the attack and investigate the possible properties. Consequently we conclude that any possible variants of the SQUARE attack do not concern the security of the full specification of the MUGI pseudorandom number generator.

The SQUARE attack against a block cipher is basically a chosen-plaintext attack where an attacker chooses a number of related plaintext blocks each of which is typically differentiated only in a byte or a word. Because of the saturation at the input of a non-linear function, the attacker can expect to control the intermediate values in 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 round key and incorrect ones.

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

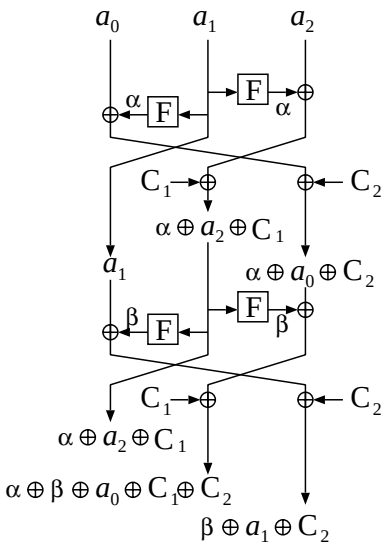

Figure 4: Two-round iteration

Related-key attacks: At first, we define the model of the attack. We assume that the attacker does not know the key value. To mount the saturation property, the attacker can *run* a number of key initializations, whose keys are differentiated only in a part of the key value, especially in this discussion we will concentrate on a word-differentiated key-dependent runs. 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 group 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 most flexibility for setting the initial state values. Let Λ the property of a intermediate word such that in each run the concerning word has different value, i.e., the word is saturated. Let O to be the property that for all runs the value is constant. Also we introduce the most weakest property "balanced" denoted by Φ that means 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 Λ,

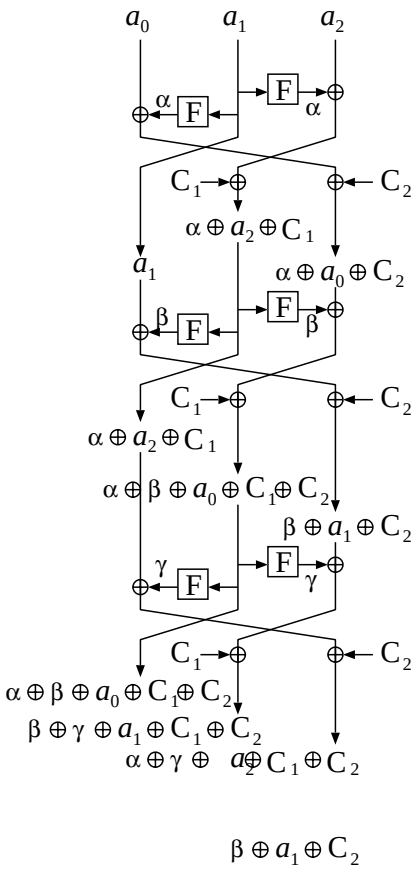

Figure 5: Three-round iteration

O, and Φ 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 to inject the saturation is the word that affects other word the last. We analyze the case of $(a_0, a_1, a_2) \xrightarrow{p} (\Lambda, O, O)$. Remember the output of the tth round is denoted by $(a_0^{(t)}, a_1^{(t)}, a_2^{(t)})$. We simply trace the property and show the results in Table (7). Hence, the initial values of the buffer b_i have the following properties depending on the

Table 7: The word properties in each intermediate values

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

index i:

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

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 value and single F-function evaluation (see the discussion below concerning to non-linear buffer relation). However, thanks to the subsequent randomization after IV injection, this property must be destroyed until the output sequence is generated. Therefore we think the related-key attack based on the SQUARE attack is inapplicable.

Chosen IV attacks: This attack may be more practical than the above related-key cryptanalysis. However, the IV 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 IV.

3.5.3 Non-linear buffer relation

The initial buffer is generated only by the secret key. The key setup algorithm generate each initial unit per a round. Since the round function of the key

setup is far from the random permutation, there are relations between initial buffer values. We describe the relations between buffer values in this part of the document.

Since the round function does not take the buffer feedback, two F-functions in each round takes the same input. Therefore, while the calculation of the ρ-function is lighter, the linear relation gets much simpler. We note that several buffer units can be expressed with a single F-functions and linear sums with other buffer unit. Here we show the table for a simple example of buffer relations:

$$\begin{aligned} b_{15} &= a_1, \\ b_{14} &= a_2 \oplus F(b_{15}, 0) \oplus C_1, \\ b_{13} &= a_0 \oplus a_2 \oplus b_{14} \oplus F(B_{14}, 0) \oplus C_2, \\ b_i &= a_0 \oplus a_1 \oplus a_2 \oplus b_{i+1} \oplus b_{i+2} \oplus F(b_{i+1}, 0) \oplus C_2, \\ & \qquad i = 12, 11, 10, \ldots, 0. \end{aligned}$$

3.5.4 All-byte equivalence

We also note an interesting property in the Panama-like key stream generator. Due to the structure of F-function, the following preliminary property can be mentioned.

Property. All-byte equivalence in F function If the all bytes in the input of the S-box layer are the same, then, the corresponding output has the same property, i.e., all the output bytes are the same as well.

Proof

Let Ω_n be the property of a n-bit register such that all bytes in the register has the same byte value. Obviously, if the 32-bit input of S-box layer has the property Ω_{32}, so does the output. Interestingly, an MDS doesn't always have the property Ω, whereas the MDS used in MUGIand Rijndael has the same property Ω_{32}. Let (x_1, x_2, x_3, x_4) and (y_1, y_2, y_3, y_4) be the input and the output of MDS. Then,

$$\begin{aligned} y_1 &= \texttt{0x02}x_1 \oplus \texttt{0x03}x_2 \oplus \texttt{0x01}x_3 \oplus \texttt{0x01}x_4, \\ y_2 &= \texttt{0x01}x_1 \oplus \texttt{0x02}x_2 \oplus \texttt{0x03}x_3 \oplus \texttt{0x01}x_4, \end{aligned}$$

$$y_3 = \texttt{0x01}x_1 \oplus \texttt{0x01}x_2 \oplus \texttt{0x02}x_3 \oplus \texttt{0x03}x_4,$$
$$y_4 = \texttt{0x03}x_1 \oplus \texttt{0x01}x_2 \oplus \texttt{0x01}x_3 \oplus \texttt{0x02}x_4.$$

If (x_1, x_2, x_3, x_4) has the property Ω_{32} or equivalently $x_1 = x_2 = x_3 = x_4$, then we obtain $y_1 = y_2 = y_3 = y_4 = x_1$. The output of MDS has the same property Ω_{32}. ¿From this property, we develop the property to F-function's property. There are two S-box and MDS streams in the F-function. To control the property Ω for the whole F-function the 64-bit input of S-box layer must be Ω_{64}.

Using this property of F-function, we can develop the property to the general structure in some extent. Remember both of the two inputs to F-function have the property Ω_{64}, then the output has the same property. Accordingly if the all *units* (64-bit registers) in the buffer and the state has the property Ω_{64} (it is not necessary that bytes beyond the unit is the same), the non-linear function F does not change the property. Thanks to the subsequent constant XORing after two F functions, this property is immediately destroyed. Consequently because of the constant XORing we think the attack based on the property Ω is not effective to attack MUGIkey stream generator.

4 Hardware Implementability

MUGI is designed so that the algorithm can be implemented to be suitable in both software and hardware implementations. In both cases, the implementation achieves the high performance and low implementation cost. In this part of the documentation, we report software and hardware implementations of MUGI pseudorandom number generator.

4.1 Software implementation

The pseudorandom number generator MUGI adopts some primitive parts used in AES, as well as the ρ function. The other operations used in MUGI are limited only to simple logical operations, e.g., exclusive-or and bit-rotation. From these profile, MUGI is expected to achieve high performance in any kinds of implementation platforms.

In this part, we report the software implementation of MUGI on the generic processor, Intel®Pentium®III processor. Our implementations we report here are all implemented by C language. Please refer to Table 8 for the detail information of the environments where MUGI was implemented.

Table 8: Software implementation environments

Hardware	CPU	Pentium®III 800MHz
	RAM	512 Mbyte
Software	OS	Microsoft®Windows®2000
	Compiler	Microsoft®Visual C++ 6.0
	Language	ANSI C
	Optimizing option	Speed

The implementation cost required for the environments specified in Table 8 is also shown in Table 9.

Table 9: Memory consumption

Code size		619 steps
work area	initialization	4.6 Kbyte
	output generation	4.2 Kbyte

The code size is the number of lines in the source code that excludes the blank and comment lines. The workarea was also measured without the memory used to store the plaintext and the ciphertext.

In this environment, MUGI can achieve the performance of 294 Mbps. In Table 10, the figure is shown in clock/byte. The resultant speed is based on the measurements on the duration for 1.2 million iteration of the update function without outputs, i.e. we neglect the delay required for memory access. The initialization takes about 15000 clocks.

Since the fundamental data size in MUGI is 64 bit, further performance advantage can be expected in the software implementation on 64-bit processors.

Table 10: Software performance

Initialization	15029 clock
Output sequence	21.8 clock/byte

4.2 Hardware implementation

In this section, we investigate the hardware implementation of the MUGI pseudorandom number generator. Toward the hardware evaluation, we focused especially on these two implementation profiles.

(1) Speed optimized circuit

(2) Gate size optimized circuit

We summarize the discussion and results of the hardware implementations. We used the hardware evaluation tool, Synopsis Design Compiler, with the cell library Hitachi ASIC HG73C (0.35μ m CMOS). In the speed optimized circuit, we evaluated the hardware can achieve the performance 2.9 Gbps with the size, 26 Kgate.

4.2.1 Speed optimized circuit

In the circuit design optimized in the performance, the applicability of the parallel computations are our most concerns. We analysed the parallelizability in detail. MUGI can be seen as an example of the PKSG, which consists of the internal state and the state-updating function. The output sequence is generated out of the current internal state. The internal state is updated according to the state-update function that defines the next state. Generation of the internal state at the time $t + 1$ requires the previous state at the time t. Therefore, implementation of plural update function in parallel will not contribute very much for speeding up one sequence generation.

We investigate the circuit with all the primitive operations of the state-update function of MUGI, in which we try to make the circuit have as many parts in parallel as possible.

We show the block diagram of the considered hardware structure of whole MUGI circuit in Figure 6. It consists of several blocks of logical circuits: the

registers K and I for the key and the initial vector, the state register a, buffer register b, the key-input processing block $ini1$, the IV-input processing block $init2$, the ρ-function, λ function block, and the logic-control block.

The control block operates each process block in response to the external signal. Each process block operates independently and in parallel. The output for a round is generated every clock signal.

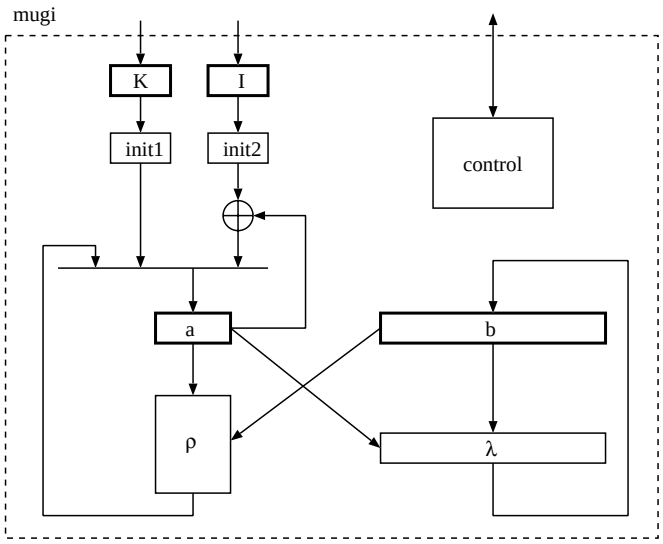

Figure 6: MUGI hardware block diagram

The block diagram of the ρ-function block is drawn in Figure 7. The ρ-function optimized for the performance consists of two F-functions and 64-bit exclusive-or operation. Each F-function includes three parts, namely, the block with eight S-boxes, the linear transformation block (MDS), and eight 8-bit exclusive or. The circuit of ρ-function can calculate the necessary date for one unit output generation within a clock-cycle.

The resultant implementation following the above strategy is summarized in Table 11.

Table 11 shows the size of each module. MUGI(whole structure) is the resultant overall structure with all modules. Because of the optimization by the hardware evaluation tool, the total gate count gets smaller than the sum of every component. When we link all the modules, the tool also optimized

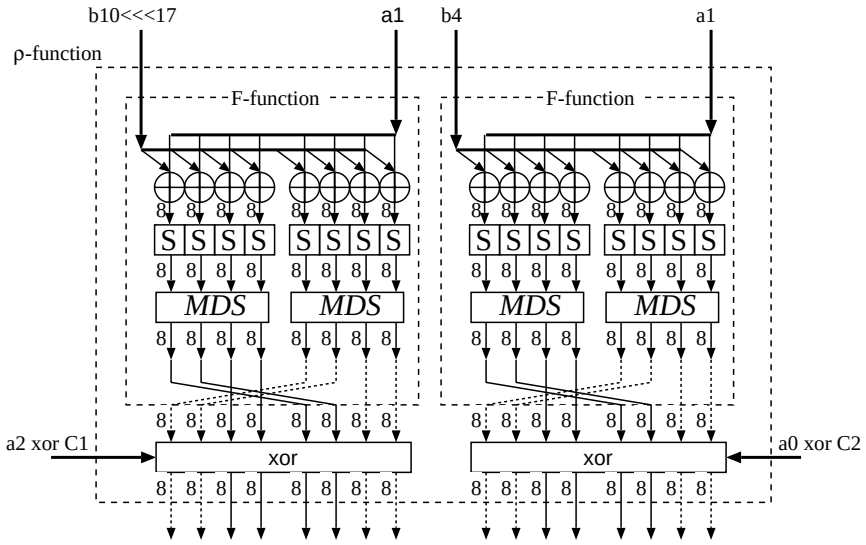

Figure 7: ρ-function structure(speed optimized)

Table 11: Gate size(speed optimized)

module	incl. lower modules gate size	module stand alone gate size
mugi(whole structure)	**26068**	**26068**
mugi	28918	14034
control	133	133
ρ	12562	360
F function	6101	128
MDS	307	307
S-box	670	670
init	181	181
λ	1826	1826

the gate size.

In this implementation, the throughput can achieve 2922 Mbps operating at 45.7 MHz. This throughput is for the whole circuit of MUGI. In addition, the initialization requires 1095 ns.

4.2.2 Gate count optimization

In the design of the gate-count optimized circuit, we analyse the structure in order for parts to share circuits of the same functionality. This decreases the total gate count. We start from the previous speed optimized circuit. In the overall block diagram of MUGI depicted in Figure 6, we find the similar functionality between $init1$ and $init2$, the key-input process block and the IV-input process block. From this observation, the circuit saves two 64-bit exclusive-ors. On the other hand, if we do so, the input of the concatenated block must have 128-bit selector to switch the key register K and the IV-register I. Taking account of those two gate saving and additional gate, the concatenation of $init1$ and $init2$ does not contribute gate saving very much.

Observing the structure of ρ-function in Figure 7, it contains 16 S-boxes and four linear transformations. We try to reduce the size of the circuit, shrinking these blocks. More specifically, we implement one linear transformation and four S-boxes instead of four and sixteen, respectively. We show the block diagram of the new construction in Figure 8. The ρ-function block shown in Figure 8 requires four clocks to calculate the data necessary for one stage output sequence, which makes the throughput of this circuit about one quarter of that of the speed optimized circuit. The input value $a1$ for ρ-function changes depending on the output of the ρ-function. To avoid $a1$'s changing, the register is put so that the $a1$ is stored. Consequently, the three-layer pipelining is applicable to this circuit, which makes the virtual latency decrease. In total the throughput is improved.

As the result of the above implementation, we show the estimation of the hardware in Table 12.

Following the above strategy, the throughput is estimated to be 676 Mbps operated at 42.3 MHz that is without three-layer pipelining. In addition, the initialization requires 4590 ns. Although we have not evaluated the pipelined circuit, our rough estimation of three-layer pipelining expects that the additional 1 Kgate may be required so that the throughput can be 2025 Mbps at

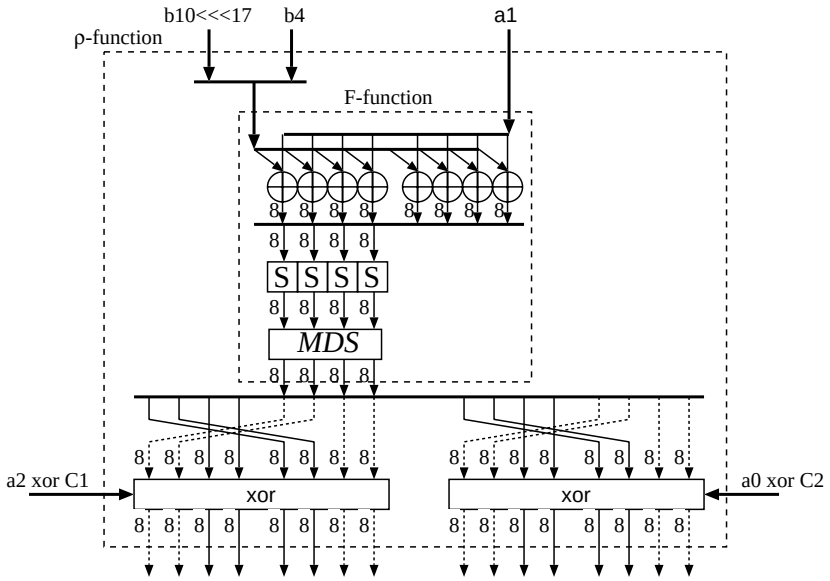

Figure 8: The structure of the ρ-function(gate-size optimized)

Table 12: Gate size(gate count optimization)

Module	incl. lower modules Gate size	Module stand alone Gate size
mugi(total)	**18019**	**18019**
mugi	20702	14489
control	250	250
ρ	3774	572
F function	3202	215
MDS	307	307
S-box	670	670
init	181	181
λ	1826	1826

126.6 MHz and the initialization requires 1531 ns.

4.2.3 Summary

The summary of the hardware implementation evaluated by the designers is shown in Table 13.

Table 13: The summary of the hardware implementations

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)

References

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

[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*, 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*, LNCS 1267, pp.149-165, Springer-Verlag, 1997.

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

[FIPS] "FIPS 140-1, Security Requirements for Cryptographic Modules," *Federal Information Processing Standard (FIPS)*, Publication 140-1, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., January, 1994, available at `http://www.itl.nist.gov/fipspubs/index.htm`.

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

[Kn81] D. Knuth, *The Art of Computer Programming Volume II (2nd ed.)*, Addison-Wesley, 1981.

[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, Proceedings Eurocrypt'93*, Springer-Verlag, LNCS 765, pp.159-169, 1994.

[MOV97] J. Menezes, C. van Oorschot, A. Vanstone, *HANDBOOK of APPLIED CRYPTOGRAPHY*, CRC Press, 1997.

[Ru86] R. A. Rueppel, *Analysis and Design of Stream Ciphers*, Springer-Verlag, 1986.

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

[WFT01] D. Watanabe, S. Furuya, K. Takaragi, "The design of key stream generator using F-function of a block cipher," SCIS 2001-6A-4, 2001 (*in Japanese*).

[WFST01a] D. Watanabe, S. Furuya, Y. Seto, K. Takaragi, "A Keystream Generator Suitable for Software Implementation," ISEC 2001-8, 2001 (*in Japanese*).

[WFST01b] D. Watanabe, S. Furuya, Y. Seto, K. Takaragi, "The correlation of the output sequence generated by the PANAMA-like keystream generator," ISEC 2001-57, 2001 (*in Japanese*).

[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`.

A Linear path of ρ

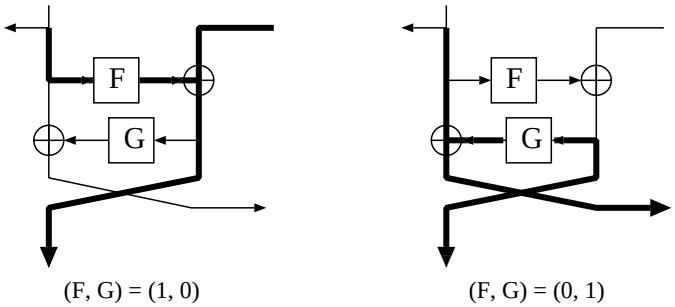

Figure 9: Linear path of ρ (1)

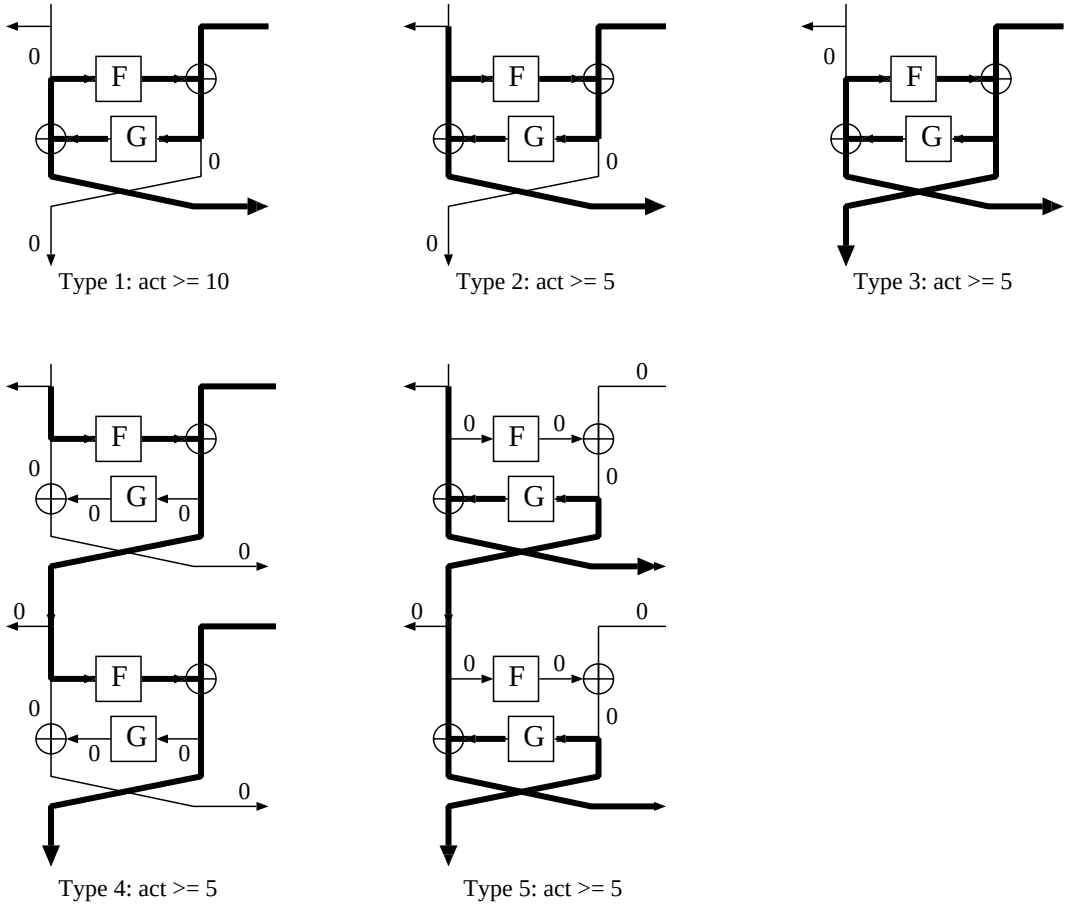

Figure 10: Linear path of ρ (2)