Evaluation of the MUGI Pseudo-Random Number Generator

July 31, 2002

Information Security Research Centre
Queensland University of Technology

Ed Dawson, Gary Carter, Helen Gustafson,
Matt Henricksen, Bill Millan and Leonie Simpson.

1 Executive Summary

This is a report on the analysis of the stream cipher MUGI [WFHP02]. For this algorithm the evaluators have

(i) investigated the security of MUGI against various attacks;
(ii) evaluated basic cryptographic properties;
(iii) evaluated statistical properties;
(iv) examined the algebraic structure of MUGI;
(v) surveyed the speed in software;
(vi) compared MUGI with the PANAMA and MULTI-S01 stream ciphers.

MUGI is a stream cipher that at each iteration outputs 64 bits from a keystream generator for its keystream.

MUGI is loosely designed on the stream cipher PANAMA but has significant differences, which will be highlighted. The evaluators have attempted to cryptanalyse MUGI with not only all the standard attacks on stream ciphers, but also using differential and linear cryptanalysis, which are more usually associated with the cryptanalysis of block ciphers. At this time none of these attacks have proved successful.

A full range of standard statistical tests has been carried out on MUGI and no potentially threatening statistical weaknesses have been found.

The evaluators conducted extensive analysis on the algebraic interaction of the stages in both the state and buffer and while many interesting relationships have been found, no exploitable weakness has as yet been uncovered. We believe this is the area where potential weaknesses may be found and from where an attack is most likely to arise. The cipher is new, and more extensive evaluation is required before exploitable weaknesses can be confidently discounted.

MUGI is reasonably fast in software due to its simplicity. The evaluators did not develop a hardware implementation so no figures are presented. The cipher is amenable to parallelisation so we speculate that it is fast in hardware.

A comparison of MUGI to the ciphers PANAMA and MULTI-S01 is provided. The evaluators believe that PANAMA has some features that make it superior to MUGI, while on the other hand MUGI does not contain the major flaw identified in the integrity check mechanism of MULTI-S01, as MUGI is to be used for data confidentiality and not data integrity.

2 Description of MUGI

An overview of the operation of MUGI is presented in this section. We examine MUGI, taking the designer's view and the generalized PANAMA-like keystream generator, but also more specifically as a keystream generator based upon a single 19 stage Non-linear Feedback Shift Register (NFSR).

2.1 General Description

MUGI is a Pseudo Random Number Generator (PRNG) designed for use as a stream cipher. Its structure is based on the PRNG, PANAMA [DC98] that can be used as either a stream cipher or hash function. A schematic that is a generalisation of the structure in both PANAMA and MUGI is presented in Figure 1 below.

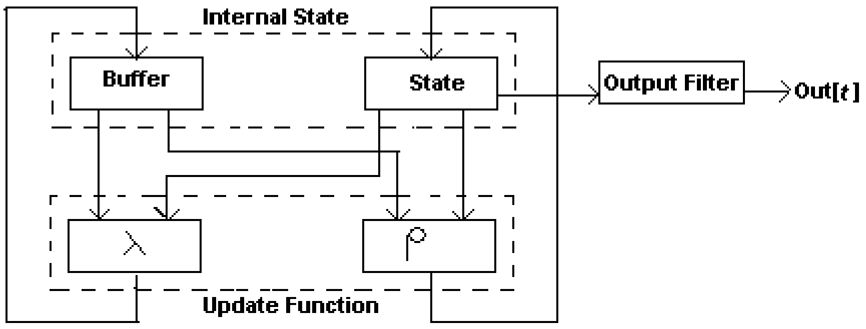


Figure 1 - The Scheme of a PANAMA-like Keystream Generator

2.1.1 MUGI: A PANAMA-like Keystream Generator

In MUGI, the internal state consists of a buffer b that is a 16-stage Linear Feedback Shift Register (LFSR), and a state a that is a 3-stage NFSR. Each stage of the buffer and state holds a 64-bit value, referred to as a *unit*. The update function of MUGI consists of a linear function λ and a non-linear function ρ that updates the buffer and state respectively. The output filter produces 64 bits of the output of state a at each iteration.

Denoting stage i ($0 \le i \le 15$) of the buffer as b_i and stage j ($0 \le j \le 2$) of the state as a_j, the details of function λ are as follows:

$$\begin{aligned}
b_i[t+1] &= b_{i-1}[t] \; (i \neq 0,4,10) \\
b_0[t+1] &= b_{15}[t] \oplus a_0[t] \\
b_4[t+1] &= b_3[t] \oplus b_7[t] \\
b_{10}[t+1] &= b_9[t] \oplus (b_{13}[t] <<< 32)
\end{aligned}$$

where b_i [t+1] and a_i[t+1] are the content of stage i of buffer b and respectively state a after the completion of t iterations of MUGI and (M < < < k) indicates a k-bit rotation to the left, of M.

Similarly, the details of function ρ are as follows:

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

where C_1 and C_2 are known constants and F is a function that uses the components of the round function of the AES [DR98].

2.1.2 MUGI: a 19-stage Non-linear Feedback Shift Register

Because of the interaction of both the state and the buffer it is possible to view MUGI as a 19-stage NFSR as in Figure 2 below. For the evaluators, this view sometimes gives a clearer picture of the interactions between buffer and state and makes it easier to trace the path of the contents of each stage during operation.

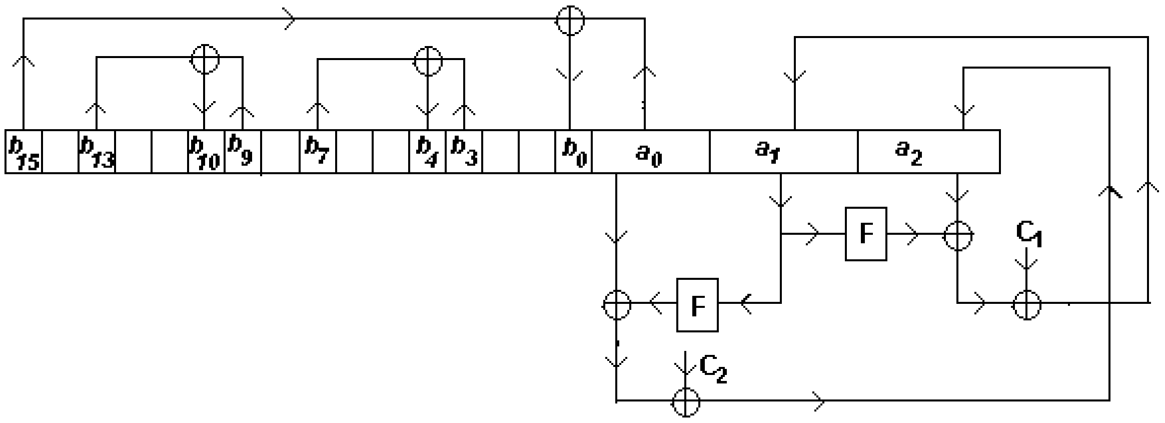


Figure 2 - Alternative View of MUGI

2.2 The Operation of MUGI

MUGI has an *initialisation* phase and an *output* phase. A detailed description of both phases is given.

2.2.1 Initialisation Phase

The initialisation phase of MUGI consists of several procedures. Firstly, the 128-bit secret key, considered as two 64-bit blocks K_0 and K_1 are entered into the state a as follows:

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

where $(M >>> k)$ represents a k-bit rotation to the right of M, and C_0 is a known fixed constant.

The second procedure uses the function ρ to put key dependent data in the buffer *b* as follows:

$$b(\mathrm{K})_{15-i} \quad = \quad (\rho^{i+1}\,(a, 0))_0 \quad 0 \le i \le 15$$

where ρ^{i+1} is the (i+1)-th iteration of ρ and the input to ρ from *b* is 0 for each *i*.

The third step in this process is to update the contents of state *a* to state *a*(K). This is done as follows:

$$a(\mathrm{K}) = \rho^{16}(a, 0)$$

The fourth procedure introduces to state *a*, a 128-bit, publicly known *initialisation vector, I* as follows:

$$a(\mathrm{K}, I)_0 = a(\mathrm{K})_0 \oplus I_0$$
$$a(\mathrm{K}, I)_1 = a(\mathrm{K})_1 \oplus I_1$$
$$a(\mathrm{K}, I)_2 = a(\mathrm{K})_2 \oplus (I_0 <<< 7) \oplus (I_1 >>> 7)$$

where I_0 and I_1 are respectively the left and right halves of *I*.

This state, *a*(K, *I*), is again updated by ρ to produce:

$$a'(\mathrm{K}, I) = \rho^{16}(a(\mathrm{K}, I), 0)$$

The final procedure in the initialisation process updates *a'*(K, *I*) to produce:

$$a[1] = Update^{16}(a'(\mathrm{K}, I), b(\mathrm{K}))$$

where *Update*(X, Y) = (ρ(X,Y), λ(X, Y)).

2.2.2 Output Phase

Each application of the *Update* function produces new values for a_0, a_1 and a_2. The 64-bit block a_2 is selected as the output and XOR'd with the plaintext when MUGI is used in stream cipher mode.

3 Structural Analysis of MUGI Components

MUGI has several components. In this section a detailed description and analysis of these components is given.

3.1 Non-linear Component of MUGI

The nonlinearity of MUGI rests entirely with the function F in Figure 3. F has both linear and non-linear components. The non-linear component consists of eight identical S-boxes each being exactly the same as the S-box used in the AES. The input to F is obtained by XOR'ing 64-bit a_1 with a 64-bit value from the buffer. Each S-box in F takes an 8-bit F-input and produces an 8-bit output. F has two further components both of which are linear. The output of the first four S-boxes is input to a Maximum Distance Separable(MDS) matrix, the same one as appears in the AES. The output of the second four S-boxes is also passed to the same MDS matrix. The final output of F is obtained from the output of the MDS matrices by byte shuffling. Figure 3 illustrates the function F.

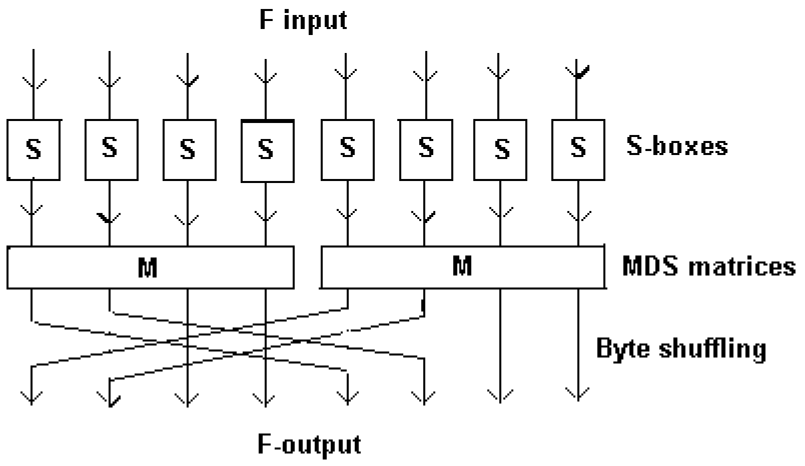


Figure 3 - Outline of MUGI F Function

3.2 Algebraic Analysis of MUGI Components and Operations

The MUGI stream cipher can be studied as the interaction between two finite state machines (FSM's), following the PANAMA-like design paradigm. In MUGI, the buffer consists of 16 64-bit units. The non-linear section of MUGI has 3 64-bit units, which are called the non-linear state. In this section expressions are derived that describe the algebraic behaviour of these parts of MUGI, with resolution at the unit level. MUGI could also be described by computation graph analysis at the 32-bit word, the 16-bit word and at the byte level, but such interesting phenomena at the unit level exist, that these other, more detailed, analyses are left to future work.

Let the values held by the buffer units at time t be denoted $b_{[0,\ldots,15]}[t]$ and let the non-linear state be similarly denoted as the triple $a_0[t]$, $a_1[t]$ and $a_2[t]$. The two FSM's interact via b_4, b_{10} and a_0, where the buffer values b_4 and b_{10} are used to affect the non-linear part in the manner of keys affecting a Feistel round of a block cipher, while the non-linear state value a_0 is used to affect the updating of the (otherwise) linear buffer in the manner of noise affecting an LFSR.

The values of $a_2[t>0]$ are given as output so they can be assumed known in a standard known plaintext scenario. The evaluators use the notation Z(t) to denote the output unit at time t.

3.2.1 Algebraic Structures in the Initialisation Process

Before a single word of output is issued, MUGI mixes keys into the state and buffer using four stages of initialisation. Stage 1 assigns the state with key material, while stage 2 mixes the state material through all 16 words of buffer. Stage 3 diffuses initialisation vectors through the state, and the final stage runs 16 rounds of the update function on both the state and the buffer, discarding all output.

In the second stage of initialisation, the fifteenth word of the buffer (b_{15}) is assigned the output $(\rho^1(a, 0))_0$, which is the value of the state variable a_0 after a single invocation of the ρ function. The ρ function simply shifts the value of a_1 to a_0. Since each buffer word is only updated once in the second stage, at the end of stage, b_{15} contains the lower 64 bits of the key K, which entered the state as a_1. Stage 3 initialisation does not touch the buffer at all, meaning that at the start of the final stage, after 32 rounds of the ρ function, half of the key material is present in the buffer in its unmixed state. An attacker has to work backwards through only 16 rounds to obtain K_1. While there is no known way of doing this faster than brute force, this is still significantly less effort than is suggested by the lengthy and complex initialisation process.

The contents of the buffer at the end of stage 2, and also at the start of stage 4 is shown in the following equations.

$$b_{15} = K_1$$
$$b_{14} = (K_0 <<< 7) \oplus (K_1 >>> 7) \oplus F(b_{15}, 0) \oplus C_1$$
$$b_{13} = K_0 \oplus (K_0 <<< 7) \oplus (K_0 >>> 7) \oplus F(b_{14}, 0) \oplus C_2$$

for $b_{12} \ldots b_0$

$$b_{\text{even}} = K_0 \oplus (K_0 <<< 7) \oplus (K_1 >>> 7) \oplus F(b_{\text{even}+1}) \oplus b_{15} \oplus b_{\text{even}+2} \oplus b_{\text{even}+1} \oplus C_1$$
$$b_{\text{odd}} = K_0 \oplus (K_0 <<< 7) \oplus (K_1 >>> 7) \oplus F(b_{\text{odd}+1}) \oplus b_{15} \oplus b_{\text{odd}+2} \oplus b_{\text{odd}+1} \oplus C_2$$

The state at the end of stage 2 is shown in the equations below.

$$a_0 = b_0$$
$$a_1 = b_2 \oplus F(b_1, 0) \oplus F(b_0, 0) \oplus C_1 \oplus C_2$$
$$a_2 = b_1 \oplus F(b_0, 0) \oplus C_2$$

In stage four, the key material K_1 quickly becomes mixed with other buffer and state. In round one of stage four, the key material is mixed with the state variable $a_0[0]$. By round five, it becomes entangled with other buffer material in the form of $b_4[5]= b_{15}[0]\oplus a_0[0]\oplus b_3[0]\oplus b_7[0]$.

There is a peculiarity in this stage of initialisation, in that the tap on b_{10} rotates one variable by half the length of the buffer word. When material passes through b_{10}, it acquires material from b_{13} which is rotated once. The material in b_{10} makes its way to b_{13}, and is again rotated for use at the b_{10} tap. This second rotations by 32 bits is the inverse of the first. While this doesn't present a security problem, it provides no additional security at the cost of a few extra clock cycles (word-swapping on 32-bit architectures, where registers can be scarce, or inefficient variable rotation on some 64-bit architectures).

Following initialisation, the first word of output contains the key material K_1, but by now it is thoroughly mixed in the form of $a_0[15]\oplus F(a_1[15], (b_{11}[0]\oplus a_0[4]\oplus b_{15}[0]\oplus a_0[0]\oplus b_3[0] \oplus b_7[0]) <<< 17) \oplus C_2$

3.2.2 Algebraic Structures in the Buffer

Let R32(x) perform a 32-bit rotation on the 64-bit value x. This will be used for convenience to replace $(x <<< 32)$ and $(x >>> 32)$. Of course $R32(R32(x))=x$ and this fact is used later in our analysis. Also $x\oplus R32(x)$ has only 32 effective unknown bits, as the R32() operation is also a swap of the two 32-bit words. Hence $x\oplus R32(x)$ has the same word twice, for any x.

The buffer update operation can be expressed by the following equations.

$$\begin{aligned}
b_0[t]&=a_0[t-1] \oplus b_{15}[t-1]\\
b_1[t]&=b_0[t-1]\\
b_2[t]&=b_1[t-1]\\
b_3[t]&=b_2[t-1]\\
b_4[t]&=b_3[t-1] \oplus b_7[t-1]\\
b_5[t]&=b_4[t-1]\\
b_6[t]&=b_5[t-1]\\
b_7[t]&=b_6[t-1]\\
b_8[t]&=b_7[t-1]\\
b_9[t]&=b_8[t-1]\\
b_{10}[t]&=b_9[t-1] \oplus R32(b_{13}[t-1])\\
b_{11}[t]&=b_{10}[t-1]\\
b_{12}[t]&=b_{11}[t-1]\\
b_{13}[t]&=b_{12}[t-1]\\
b_{14}[t]&=b_{13}[t-1]\\
b_{15}[t]&=b_{14}[t-1]
\end{aligned}$$

These express the buffer values in terms of other buffer values at the previous times, and are useful for implementation. In contrast, the evaluator’s task is to express the buffer contents in terms of some values in future times. With this motivation, reversed buffer operation can be considered, in which the above equations are reversed. Given a_0 is known, it follows that

$$b_3[t]=b_4[t+1] \oplus b_7[t]$$
$$b_9[t]=b_{10}[t+1] \oplus R32(b_{13}[t])$$
$$b_{15}[t]=b_0[t+1] \oplus a_0[t]$$

and

$$b_n[t]=b_{n-1}[t+1] \qquad \text{for other values of n.}$$

It is now possible to derive some new equations from these that are useful in cryptanalysis. Firstly, consider that by substituting several times in consecutive equations in the above lists expressions across any number of iterations can be achieved, and this opens the way to define each buffer location in terms of a_0, b_4 and b_{10} only. For example the following equations:

$$b_9[t]=b_4[t-5]$$
$$b_{13}[t]=b_{10}[t-3]$$

can be used to obtain an interesting result that expresses each b_4 value in terms of future b_{10} values only.

Now consider the equation

$$b_{10}[t]=b_9[t-1] \oplus R32(b_{13}[t-1])$$

and substitute $b_4[t-6]$ for $b_9[t-1]$, and $b_{10}[t-4]$ for $b_{13}[t-1]$, giving

$$b_{10}[t]=b_4[t-6] \oplus R32(b_{10}[t-4]).$$

Now re-arrange to make b_4 the subject and let t=t+6 yielding

$$b_4[t]=b_{10}[t+6] \oplus R32(b_{10}[t+2])\ldots\ldots\ldots\ldots\ldots\ldots1$$

All this information comes from an analysis of the buffer alone. This equation is true for all t and shows that the entire *b* sequence is defined by a simple linear recursion on the b_{10} sequence. Each bit of every $b_4[t]$ is given as the XOR of two bits from the nearby b_{10} sequence.

There are other interesting expressions to be found in the buffer. Start with

$$b_0[t]=b_4[t+4] \oplus b_7[t+3]$$

and substitute $b_0[t]=a_0[t-1]\oplus b_{15}[t-1]$, and $b_7[t+3]=b_{10}[t+6]\oplus R32(b_{13}[t+5])$ and $b_{13}[t+5]=b_{10}[t+2]$ to give

$$a_0[t-1] \oplus b_{15}[t-1] = b_4[t+4] \oplus b_{10}[t+6] \oplus R32(b_{10}[t+2])\ldots\ldots2$$

This result connects all three of the data streams that link the linear with the non-linear stage of MUGI. Re-arranging the terms (since all operations are XOR and hence associative) and uniformly altering all t values (by shifting the equation to another time) gives many other equivalent expressions.

Combining equations 1 and 2 by substituting for b_4 gives a new result that removes b_4 from the expression. The following relates the b_{10} stream with a single value of a_0. It expresses a $b_{10}[t]$ in terms of future values.

$$b_{10}[t]=a_0[t+5] \oplus b_{10}[t+16] \oplus b_{10}[t+12] \oplus R32(b_{10}[t+12]) \oplus R32(b_{10}[t+8]) \ldots .3$$

Observe that equation 3 has x⊕R32(x), which has only 32 effective bits. In an attack one might guess these 32 bits and thus remove b_{10} from the equation.

In addition notice that equation 3 can be re-written with $b_{10}[t] \oplus b_{10}[t+16]$ as the subject. In this new expression all the right hand side terms involve times between these two extremes of t and t+16.

It is possible to derive similar 16 round difference expressions for all of the different buffer units, and with all the right hand side terms selected from the a_0, b_4 and b_{10} streams. There are three different general forms for these expressions, corresponding to the three natural parts of the buffer. The derivations of all these separate expressions using the basic buffer equations are very tedious, so it appears in Appendix A. There are three general forms as follows.

For n ={0,1,2,3} we have

$$\begin{aligned} b_n[t] \oplus b_n[t+16] =& a_0[t+15-n] \oplus b_4[t+4-n] \oplus b_{10}[t+6-n] \oplus \\ & R32\,(a_0[t+7-n] \oplus b_4[t+12-n] \oplus b_{10}[t+14-n]) \end{aligned}$$

For n ={4,5,6,7,8,9} we have

$$\begin{aligned} b_n[t] \oplus b_n[t+16] =\quad & a_0[t+19-n] \oplus b_4[t+24-n] \oplus b_{10}[t+10-n] \oplus \\ & R32\,(a_0[t+11-n] \oplus b_4[t+16-n] \oplus b_{10}[t+18-n]) \end{aligned}$$

For n ={10,11,12,13,14,15} we have

$$\begin{aligned} b_n[t] \oplus b_n[t+16] =\quad & a_0[t+15-n] \oplus b_4[t+20-n] \oplus b_{10}[t+22-n] \oplus \\ & R32\,(a_0[t+23-n] \oplus b_4[t+28-n] \oplus b_{10}[t+30-n]) \end{aligned}$$

3.2.3 Algebraic Structures in the Non-linear Part of MUGI

Considering the non-linear part of MUGI first note that the a_1 value is used twice as input to the same 64-bit bijection F, with the addition of different parts of the current buffer by XOR at the input to F. Also each a_1 value is used in the next iteration as the a_0 value that affects the buffer, so clearly the a_1 values are important. Thus, the task is to define these buffer values in terms of the known output values a_2=Z.

The non-linear part is defined as

$a_0[t]=a_1[t-1]$…………………………………………………..4
$a_1[t]=C_1 \oplus a_2[t-1] \oplus F(a_1[t-1] \oplus b_4[t-1])$………… …………….5
$a_2[t]=C_2 \oplus a_0[t-1] \oplus F(a_1[t-1] \oplus R17(b_{10}[t-1]))$……………………6

Substituting equation 4 into both 5 and 6 to remove all a_1 values, leaves a_0, a_2 and buffer values only. Note that C_1 and C_2 are known constants based on the decimal expansion of irrational numbers.

$$a_0[t-1]=C_1 \oplus a_2[t-1] \oplus F(a_0[t]\oplus b_4[t-1])$$

By applying these expressions (and time shifted versions) we can derive interesting equations relating known output a2=Z values with the a0, b4, and b10 sequences. For example it can be shown that

$R17(b10[t]) + b4[t] = F^{-1}(a0[t+1]+Z[t+2]+c2) + F^{-1}(a[t+3]+Z[t+1]+c1)$………7

This expression uses the reduced F() function that is considered in detail in Section 5.1. The F() function is an SPN style bijection using the 8*8 s-box and 32-bit linear transform from the AES. The inverse function $F^{-1}()$ is well defined from the the AES specifications. Even though the inverse operation is not used in MUGI, we are able to consider it in cryptanalysis.

We note that in equation 7 we can assume knowledge of the Z values and also the C1 and C2 values are public constants. Equation 7 was derived by considering two separate expressions for a1[t] and equating them thus removing a1[t] from the overall expression.

An alternative derivation based on equating a0[t+2] leads to this expression:

$C1+C2+Z[t]+Z[t+3] = F(a0[t+1]+b4[t]) + F(a0[t+3]+R17(b10[t+2]))$………..…8

While being equivalent to equation 7, this version may be more useful as it separates the known values from the unknown.

These algebraic techniques can be used for equating Z[t] values also, leading to different results. It can be shown that

$C1+C2=F(a0[t+1]+R17(b10[t]))+F(a0[t+2]+b4[t+1])+a0[t]+a0[t+3]$…………..9

We note that this expression is true for all times t and it does not depend on any part of the output sequence. Thus equation 9 represents an inherent property of the non-linear part of MUGI. The function of the 6 internal values
a0[t],a0[t+1],a0[t+2],a0[t+3], b4[t+1], and b10[t] is always the same known constant for all times t. Thus this set of six values always has some redundancy.

4. Keystream Properties

For keystream sequences to be used in stream ciphers that provide cryptographic security, the keystream must possess certain basic properties. These include a large period, large linear complexity and white-noise statistics.

Experiments were conducted by the reviewers using the CRYPT-X package. This is a statistical package previously designed by the reviewers for analysing encryption algorithms. The relevant pages from the CRYPT-X manual have been included in the manual. The experimental results are included below.

Statistical analysis was conducted by the evaluators of MUGI on two different keystream types: one with a variable initial key (K) and a fixed initialisation vector (IV), and the other with a fixed initial key (K) and a variable initialisation vector (IV).

4. 1. Period

Estimating the period of the MUGI output sequences is difficult due to the use of the nonlinear update function for state a. The theory regarding the period of the sequences produced by linear feedback shift registers (LFSRs) is well known. For autonomous LFSRs of length L with primitive feedback polynomials, the output sequences attain the maximal length of 2^L-1, provided the initial state of the LFSR is nonzero. For example, if the LFSR used for the buffer was autonomous (no input from state a) and had primitive feedback polynomial, then the period of the output sequence of b would be $2^{16x64}-1 = 2^{1024}-1$. However, for nonlinear feedback shift registers (NSR) no such theory exists.

An upper bound on the period of sequences produced by MUGI is to take the view that MUGI is a finite state machine (FSM), with a single NSR of nineteen 64-bit units, as illustrated in Figure 2. At some point in time, the internal state of this FSM will return to an earlier internal state. This will define the period of the output sequence. Thus, the maximum period is $2^{19x64}=2^{1216}$.

Of course, for cryptographic applications, we are more interested in a lower bound on the period. However, the use of the nonlinear update function prevents us from applying known shift register theory to determine whether the FSM will cycle through all states before returning to an initial state, or whether shorter state cycles will occur. The buffer uses linear feedback, but is non-autonomous. The initialisation of the buffer with the 128 bit key K provides 2^{128} initial states for the buffer. The effect of XORing the contents of a_0 with the contents of b_{15} in the linear update function is not determined. However, it is anticipated that this will, in general, lengthen the period of the sequence produced by the buffer, so with high probability, the period of the buffer sequence will exceed 2^{128}. This may be a conjectured lower bound on the period of the keystream sequence.

4. 2. Linear Complexity

The linear complexity of a binary sequence is defined to be the length of the shortest binary LFSR which can reproduce the sequence. The Berlekamp-Massey algorithm can construct such an LFSR from a known keystream segment of length only twice the linear complexity. Thus, for cryptographic applications, a large linear complexity is required. The MUGI design, with the interdependence of buffer b and state a complicates any calculation of the linear complexity. Considering MUGI as a finite state machine, the application of the nonlinear function should ensure that the linear complexity greatly exceeds the length of FSM. That is, the linear complexity should be well in excess of 19x64 = 1216. We rely on empirical evidence to estimate the linear complexity.

Experimental results, which are included below for linear complexity and statistical analysis, were conducted by the reviewers using the CRYPT-X package. This is a statistical package which was previously designed by the reviewers for analysing encryption algorithms. The relevant pages from the CRYPT-X manual have been included in the manual.

Statistical analysis was conducted by the evaluators of MUGI on two different keystream types: one with a variable initial key (K) and a fixed initialisation vector (IV), and the other with a fixed initial key (K) and a variable initialisation vector (IV).

The linear complexity checks for the minimum amount of knowledge required to reconstruct the whole stream using a linear feed back shift register. It is difficult to determine the linear complexity of a sequence from MUGI. In order to obtain empirical evidence for the linear complexity and linear complexity profile the tests from the CRYPT-X package were applied.

The linear complexity tests were applied to five MUGI keystreams (variable key with fixed IV) of length 819,200 bits (this small number was due to the time taken for the test). Results showed linear complexity values close to that expected for random data (i.e. half the bit-stream length = 409,600).

Test	***Key 1***	***Key 2***	***Key 3***	***Key 4***	***Key 5***
Linear Complexity	409,600	409,600	409,600	409,600	409,599
LC p-value	0.5	0.5	0.5	0.5	0.1659
Linear Complexity Profile:					
Jumps p-value	0.2880	0.6047	0.6392	0.7753	0.2722
Jump Size p-value	0.6503	0.4123	0.4517	0.5357	0.2203

The linear complexity test was also applied to 100 MUGI keystreams of length 10^5 bits (for both variable key with fixed IV, and fixed key with variable IV), and these results also showed linear complexity values close to that expected for random data (i.e. half the bit-stream length). For more detailed results see Appendix C. The results for the linear complexity profile indicate that, as the bit-stream increases in length, the changes in linear complexity maintain the expected value of half the stream length. These results support the randomness of the keystream output from MUGI, based on linear complexity, such that the whole bit-stream is required to re-construct the stream itself, thus giving an attacker no advantage in being able to create the bit-stream with a smaller number of output bits.

4. 3. CRYPT-X Statistical Tests

The statistical analysis applied to MUGI keystreams are the tests explained in the CRYPT-X package (see Appendix C), namely the frequency, binary derivative, change point, subblock, runs distribution (+ longest run), sequence complexity and linear complexity tests. The tests are based on the hypothesis that the measure obtained from the output stream supports randomness. The p-values obtained from the tests represent the probability that such sample result (or a less random one) would be obtained if the algorithm produces a random stream. Very small p-values would support non-randomness.

The first five tests were applied to one hundred different MUGI keystreams of 1 Megabyte (2^{23} = 8 388 608 bits), and the two complexity tests were applied to the first 10^5 bits of these keystreams (due to the amount of time required for these tests).

The subblock test was applied to the MUGI keystreams by dividing the bit-stream into non-overlapping subblocks of length 4 and 8 bits.

The frequency test was also applied to bit positions 1 to 64 in non-overlapping subblocks of 64 bits on the one hundred MUGI keystreams. The number of bits in each test was $2^{23}/2^6$ = 131,072 bits.

Results of Statistical Analysis

The results give the lowest p-value for any one test as 0.0001, with the proportion of p-values falling below 0.1, 0.05 and 0.01 equal to that expected (over all the 14,800 tests applied) . This supports the randomness of the output from MUGI. For more detailed results see Appendix C. The tables of results give the number of p-values falling in the bottom 10%, 5%, and 1% for each test applied to 100 different MUGI keystreams. It is expected that approximately10 p-values should fall below 0.1, 5 below 0.5, and 1 below 0.01 from the 100 keystreams tested.

In the results of the statistical analysis of MUGI presented in [Hit01] there is recorded a run of length 31 in a sample of length 2^{21} bits. We agree that this is a rare event with a small probability. The length of the longest run, recorded by the reviewers, for the samples of 1 megabyte (2^{23} bits) was 34. Combined with the results of the runs distribution test applied from CRYPT-X package, this indicates there is no problem with the distribution of runs in samples generated by MUGI.

The sequence complexity test provides an effective method of detecting periodicity or periodic patterns in the bit-stream. In the bit-streams tested all sequence complexity values exceeded both the threshold value and the average value of sequence complexity for bit streams of length 10^5 bits. These results support that the period of the keystreams exceeds the length of the streams tested, and that there was no detection of patterns in the MUGI keystreams.

5 Analysis of MUGI

In this section possible cryptanalytic attacks on MUGI and its components are examined.

Attacks on stream ciphers are based on a number of assumptions. For synchronous binary additive stream ciphers, including MUGI, known plaintext-ciphertext pairs are simply XORed to reveal segments of keystream, z. Known plaintext attacks are based on the assumption that the entire structure of the keystream generator is known, and additionally that the cryptanalyst has access to some amount keystream produced by the generator. Generally the task of the cryptanalyst is to use the known plaintext and structure to either determine the key, or to construct an equivalent keystream generator (one which produces the same known keystreams). An alternative model is considered in [CHJ02]: if an attacker watches the output stream, can the attacker distinguish it from a truly random stream? Note that for the keystream generator MUGI, in addition to being the output of the keystream generator, a segment of known plaintext of length N also reveals the contents of a2, one of the three units of the state a, for N iterations.

Depending on the application, the cryptanalyst may have access to only a single known keystream z, or to multiple known keystreams z_j. This latter case is common where the stream cipher must be resynchronised. Generally for resynchronisation the cipher is initialised with a publicly known vector I_0, in addition to the secret key K, and rekeyed with the same secret key and a different (but possibly related) vector I_j. The cryptanalysts' task is to recover the key K given a set of $(z_{j,}\ I_j)$ pairs. Extensions of the resynchronisation attack scenarios include having multiple keystreams produced by multiple unknown (but possibly related) keys and a single known I, or even multiple keystreams produced by varying both K and I, under some assumed relationship.

To determine if the cipher used is MUGI, a linear masking attack can be performed. For the general cryptanalytic attacks (recovering the key), several methods for attacking shift-register based stream ciphers are well known. These include time/memory/data trade-off attacks, correlation attacks and divide and conquer attacks. These methods will be outlined briefly in this section, and discussed with respect to the MUGI stream cipher under the different attack scenarios in the following sections.

Section 5.1 and section 5.2 consider components of MUGI, while the remaining sections of this chapter examine attacks on MUGI as a whole. In this report, the attack scenarios which are considered are the following: single keystream and multiple keystream with fixed key and varying I.

5.1 Differential and Linear Cryptanalysis of the F function

Differential [BS93] and Linear Cryptanalysis [Mat94] are powerful generic tools that can be applied to most ciphers. In differential cryptanalysis, an input difference (often XOR) produces an output difference with a certain probability, which if large enough can often lead to the determination of the key. Similarly, linear cryptanalysis exploits the difference from .5 (called a *bias*) of the probability that a linear sum (often XOR) of input and key bits is equal to a linear sum of output bits.

The F function of MUGI is modelled on the round function of the block cipher Rijndael, now the Advanced Encryption Standard(AES). Both functions follow the Wide-Trail Strategy [DR98]. The aim of this strategy is to maximize the number of active S-boxes in consecutive rounds, meaning that the cumulative probabilities of differentials and linear approximations become prohibitive for attackers in a very small number of rounds. MUGI reuses AES' single S-box and its (32, 32) Maximum Separable Distance (MDS) matrix. The S-box is used eight times in a round and provides confusion, while the MDS is used twice and provides diffusion. The outputs of the MDS are intertwined in MUGI, in lieu of an AES ShiftRow-like operation that mixes different thirty-two bit words. Because the round-function of MUGI is very similar to that of the AES, much of the analysis for the latter should apply here.

A well-known property of the MDS used by the AES, is that the sum of active (non-zero) input bytes and output bytes is lower bounded by five. This is termed the branch number, and is the key component in realizing the Wide-Trail Strategy. The branch number of the MDS strongly influences the number of rounds that can be attacked by statistical cryptanalytic techniques like differential and linear cryptanalysis. When combined with the other diffusion elements in the AES, the MDS causes the number of active S-boxes in four rounds of AES to be lower-bounded by 25 [DR02].

The round function of MUGI is 64-bits wide, half the size of the AES round, making comparisons of the active S-box tally after four rounds difficult. One quantity that can be measured is the efficiency of the intertwined MDS outputs to provide increased diffusion as compared to two isolated MDS matrices executed in parallel.

The evaluators tested the MDS structures by looking for self-iterating patterns at the byte level. The inputs to the MDS structures map to themselves as the output of the structures after four rounds. The number of active S-boxes within each pattern were tallied and compared for both structures. The results for the MUGI MDS-structure are shown in Table 1, and the results for the isolated MDS matrices are shown in Table 2.

Table 1 - Active S-boxes in the MUGI MDS-structure

# Active S-Boxes	# MDS Sets	# Active S-Boxes	# MDS Sets
15	8	≤16	12
≤ 17	16	≤ 20	30
≤ 21	42	≤ 22	54
≤ 23	58	≤ 24	76
≤ 25	116	≤ 26	208
≤ 27	236	≤ 28	238
≤ 29	254	≤ 32	255

Table 2 - Active S-boxes in isolated MDS matrices

# Active S-Boxes	# MDS Input Sets	# Active S-Boxes	# MDS Input Sets
11	8	≤12	13
≤13	28	≤16	30
≤22	48	≤23	63
≤24	130	≤25	162
≤26	226	≤27	235
≤28	238	≤29	255

This demonstrates that swapping two of the four MDS outputs generally contributes more to diffusion than not swapping. There are cases when this is not the situation, although these are in the minority. For some inputs (such as X X 0 0 0 0 X 0, where X is an active byte), swapping causes twenty active bytes over four rounds, whereas not swapping causes twenty-four active bytes. Conversely, swapping sometimes contributes to as many as nine extra active bytes in four rounds.

Testing of the MDS-structures in this way showed that there are many mappings in which the input pattern emerges intact as the output of the structure after four rounds. These can be used as self-iterating characteristics in differential cryptanalysis provided that active S-boxes also preserve the self-mapping with high probabilities. The MUGI S-box and MDS structures work very well together at preventing self-mapping on a bit level, so that most self-iterating MDS patterns do not extend to the full round, but there are still useful mappings.

5.1.1 Differential Attack on the MUGI F Function

To engineer a successful differential attack against the F function, the probability of the differential cannot exceed 2^{-64}, since such an attack would require more than the entire codebook.

Analysis of the MUGI S-box shows that for any input difference, one output difference (DP_{Max}) has a probability of 2^{-6} and 126 differences have a probability of 2^{-7}. The remaining output differences have probability zero. Consequently, any differential cannot encounter more than $2^{-64}/2^{-6} = 11$ active S-boxes. In all likelihood, differentials will not be able to use DP_{MAX} across each S-box, reducing further the maximum number of active S-boxes to 8.

From Table 1, it can be seen that there are no MDS patterns that permit 8 s-boxes across four rounds, so no differentials can be constructed across four rounds. The branch number of the MDS guarantees that differentials across four rounds with 15 active S-boxes will pass across three rounds with no fewer than 11 active S-boxes. So differential attacks on the F-function apply to no more than two rounds.

This result has implications upon the success of a differential attack on the ρ function past two rounds. For some of the rounds in a multiple-round attack, the a_1 state variable will need to contain no active bytes. Differences held by the a_0 and a_2 state variables may need to cancel, prior to entry into the F function, a situation that rarely occurs because of the tendency of the MDS to increase the number of active bytes, rather than to decrease them. This attack is not feasible.

5.1.2 The Saturation Attack

The Saturation (a.k.a Square) attack can be applied to MUGI's F-function in much the same way as for the AES cipher. The attack deals with a set of texts and the relationships of corresponding bytes within the texts – individual values of the bytes are ignored. For MUGI, the attack uses an alpha set – a set of 256 texts that each differ in one or more bytes. The symbol "–" denotes a constant byte position that has the same value in each of the texts. The symbol "A" denotes an active byte position that has a unique value for each of the texts. The symbol "⊕" describes a balanced byte position in which the byte values of the texts sum to zero modulo 256. Finally, the symbol "?" covers the remaining cases in which the relationships between corresponding bytes in the texts are unknown.

An attack on three rounds of the MUGI F-function is shown in Figure 4. For all cases with a single active byte, the output remains the same.

Round 1	A – – – – – – –
	⇓ s-boxes
	A – – – – – – –
	⇓ MDSes
Round 2	– – A A A A – –
	⇓ s-boxes
	– – A A A A – –
	⇓ MDSes
Round 3	⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕
	⇓ s-boxes
	? ? ? ? ? ? ? ?

Figure 4 - Saturation attack on MUGI F-function

The value of the round key does not affect the saturation attack. The bijective s-boxes of the first round do not destroy the any of the relationships within the bytes, but the MDS propagates the active byte input to four positions of the output. The second round s-boxes maintain the status-quo, but the MDS change the active bytes to balanced bytes. Finally, the third-round s-boxes destroy the balanced relationships between the texts.

The attack technique involves guessing key material for the last round of the F-function, the balance of the active set having been destroyed by the time it emerges as output. The attacker decrypts to the point where, if the key guess were correct, the balance of bytes is restored. If not, the key guess is incorrect. The attack is effective because individual bytes can be guessed independently.

Because MUGI lacks a strong diffusion element between thirty-two bit words, the balance of the alpha-set is destroyed in the third round, rather than the fourth, as in the

AES. However, because MUGI combines key and input material at the start of its round, rather than the end, it is trivial to roll back the s-box and MDS operations on the fourth-round output. From then, the attacker guesses key bytes, and decrypts to the point just before s-box application to check the byte balance.

An attack on four rounds of the F-function involves guessing eight bytes of key material. The attack requirements are 512 chosen plaintexts and 512 cipher encryptions. As for the AES, the attack can be extended to five rounds for 2^{11} plaintexts and 2^{40} executions. The attack cannot be extended to six rounds because of excessive text requirements.

The authors of [Hit01] show how the attack can be generalized to the ρ function by saturating one byte of the a_0 state variable through the key. They gain two rounds "for free" before the saturated byte reaches the F-function, and are able to continue it for four rounds. They state that thirty-two applications of the ρ function upon the state is more than adequate in preventing the attack. The evaluators agree with this assessment.

5.1.3 Linear Attack on the MUGI F Function

The MUGI S-box is not as resilient against linear cryptanalysis as it is against differential cryptanalysis. Table 3 shows the number of inputs, for any masks, that gives a particular output bias. These biases can be used as the basis for linear approximations.

Table 3 - Linear biases in MUGI S-box

Bias	# Inputs
0	17
±2	48
±4	36
±6	40
±8	34
±10	24
±12	36
±14	16
±16	5

While [Hit01] claims that the maximum bias for the S-box is 2^{-6}, it is clearly 2^{-4}. It is easy to find linear approximations across a single F function round with a probability of 2^{-16}, one such approximation being 81F7676700000000→0000000001000000. The branch number of the MDS mandates that there are five active bytes in each single-round expression. Unlike differential cryptanalysis, however, in which the number of texts required for an attack is inversely proportional to the probability of the differential, linear cryptanalysis has much more demanding requirements relating the texts inversely to the *square* of the probability. For the codebook of 2^{64} texts, the approximation can incorporate no more than 8 active S-boxes. This limits linear cryptanalysis of the F function to three rounds.

The chances of a successful linear attack on the ρ function are slim for the reasons posited by the differential attack argument – one ρ round contains two calls to the F function, and cancellation of active bytes is made problematic by the properties of the MDS.

5.2 Algebraic Analysis

In section 3.2 algebraic relationships were developed which could prove an aid the cryptanalyst. Therefore the cryptanalyst's task can be seen as exploiting the initialisation process, the buffer and nonlinear stage updating equations in order to recover information about the initial value of all these units from the sequence of known a_2 values

5.2.1 Algebraic Analysis of the Initialisation Process

In section 3.2.2 it was shown that at stage 2 of the initialisation process that the buffer keeps a record of the state before known initialisation vector(*I*) material is injected into the cipher. When b_{15} and b_{14} are known, the key is known and it is trivial to extract the remaining buffer material and the state. Guessing b_{15} and b_{14} is equivalent in effort to guessing the entire 128-bit key. However, it is curious that a word contains key material "in the clear" so close to the end of the initialisation process, and that with its neighbouring word, all key material for the cipher session can be trivially recovered.

5.2.2 Algebraic Analysis of the Buffer

In section 3.2.3 the following equations were derived.

For n ={0,1,2,3} we have

$$\begin{aligned} b_n[t] \oplus b_n[t+16] = {} & a_0[t+15-n] \oplus b_4[t+4-n] \oplus b_{10}[t+6-n] \oplus \\ & R32\,(a_0[t+7-n] \oplus b_4[t+12-n] \oplus b_{10}[t+14-n]) \end{aligned}$$

For n ={4,5,6,7,8,9} we have

$$\begin{aligned} b_n[t] \oplus b_n[t+16] = {} & a_0[t+19-n] \oplus b_4[t+24-n] \oplus b_{10}[t+10-n] \oplus \\ & R32\,(a_0[t+11-n] \oplus b_4[t+16-n] \oplus b_{10}[t+18-n]) \end{aligned}$$

For n ={10,11,12,13,14,15} we have

$b_n[t] \oplus b_n[t+16] = a_0[t+15-n] \oplus b_4[t+20-n] \oplus b_{10}[t+22-n] \oplus$
$R32\ (a_0[t+23-n] \oplus b_4[t+28-n] \oplus b_{10}[t+30-n])$

Notice that the left hand side of each of the above equations is the XOR sum of the same buffer position now and 16 cycles later, for every time t This is a simple expression for the effect that 16 rounds has on the buffer. It can be seen that exactly 16 rounds are used in each active stage of the initialisation process, as well as being the natural length of the buffer itself during keystream output, so it is possible that these simple expressions may assist cryptanalysis of MUGI, although to date it has not been possible to turn them into an attack.

5.2.3 Algebraic Analysis of the Nonlinear Part of MUGI

In section 3.2.4 a linear expression was obtained for the nonlinear part of MUGI which eliminated the a_1 value. The resulting expression only involves a_0 which is related to a_2, the known output, a_2 itself and the output of the F function.

By combining equations 7, 8 and 9 for the nonlinear part with the expressions for the buffer, we may hope to discover results that have application in cryptanalysis. However, initial investigations suggest that not all intermediate values cancel out, leaving more unknowns than equations. As the secret key is 128 bits = 2 units, any interesting attack would have to be based on guessing fewer than two units. However we have not yet found any useful expressions that have fewer than two unknown values.

5.3 Attacks on MUGI

5.3.1 Linear Masking Attack Applied to MUGI

The Linear Masking Attack [CHJ01] can be applied to ciphers consisting of two repeating functions: one linear and the other nonlinear. As such, it has direct application to MUGI, where the updating function for the buffer *b* is a linear function and the updating function for the state *a* is a nonlinear function. The attack model considered is: if an attacker watches the output stream, can the attacker distinguish it from a truly random stream? If so, how much text, time and space are required? This can be viewed as a hypothesis testing problem. According to the null hypothesis, the bits observed as inputs and outputs for one round are random and independent. Alternatively, they are generated by the MUGI cipher. A decision rule is required to determine whether the null hypothesis can be rejected in favour of the alternative.

Note: this is not the usual cryptanalytic problem: given a segment of known keystream from a keystream generator for which the structure is known,

1. Recover the secret key, k or
2. construct a keystream generator which can produce the same (or very nearly) keystream sequence.

However, some of the stages of this attack are stages that may be performed as part of other attacks. They are outlined briefly below.

The attack is performed in three stages. The first stage focuses on the nonlinear function, to try to find a characteristic that distinguishes it from random (for example, a linear approximation which is more often 0 than 1). The second stage examines the linear function, to find a linear combination of the linear process which vanishes. The same combination of the known sequence would then remove the linear part of the process, leaving a linear combination of the nonlinear process. The 'characteristic' found in the first stage can then be applied. For example, if a linear approximation is found, in the third stage we have a sum of linear approximations of the nonlinear process.

In the case of MUGI, the internal state consists of the 'nonlinear' state *a* and the 'linear' state *b*. In each step, some bits of *a* are added to some bits of *b*, and some other bits of *b* are added to some bits of *a*. Then the nonlinear function is applied to *a* and the linear function is applied to *b*. Finally, some bits of a and *b* are added together to form the output of this step.

More specifically, consider the internal state of MUGI at time t. The update functions to obtain the internal state at time t+1 can be described in terms of the three stages outlined above:

1) two units of *b* (b_4[t] and b_{10}[t]<<<17) are added (XORed) (separately) to one unit of *a* (a_1[t]) resulting in $a_1[t]_1$ and $a_1[t]_2$, and one unit of *a* (a_0[t]) is added to one unit of *b* (b_{15}[t])

2) then, the nonlinear function NF(a) is applied to the resulting a:

$a_0[t+1] = a_1[t]$,
$a_1[t+1] = a_2[t] + F(a_1[t]_1) + C_1$
$a_2[t+1] = a_0[t] + F(a_1[t]_2) + C_2$

and a linear function is applied to the resulting *b*:

$b_0[t+1] = b_{15}[t]$,
$b_{i+1}[t+1] = b_I[t]$ for i = 0, 1, … 14 except i = 4, 10
$b_4[t+1] = b_3[t] + b_7[t]$
$b_{10}[t+1] = b_9[t] + (b_{13}[t+1] <<< 32)$

and finally,

3) the contents of a_2[t+1] are output as the keystream unit at time t+1.

In applying the linear masking attack, note that the MUGI keystream consists only of output from the nonlinear function. Contents of the linear state are not added directly into the output. Hence the major task is in identifying a characteristic such as a linear approximation of the nonlinear function which is used to form the output.

Note that in the nonlinear function NF(a), the nonlinearity is concentrated in the F operation. The other operations consist of unit shifts of the register, and XORs.

In order to apply this attack, we need to find some linear combination of inputs and outputs of the nonlinear function for which some bias exists.

This attack is not considered a threat to MUGI as this point in time.

5.3.2 Key Recovery Attacks

Exhaustive key search. Try each of the possible keys until a key is found which produces the required keystream.

Time/Memory/Data tradeoff attacks on stream ciphers (see [Bab95] and [Gol97]) are known plaintext attacks, conducted in two phases. In the first, preprocessing phase, the attacker tries different keys and stores a prefix of the output sequence in a table. In the second (realtime) phase, the attacker has a segment of known keystream and uses the precomputed table, with the objective of recovering the internal state at a known time.

Let S, M, T, P and D denote the cardinality of the internal state space, the memory (in binary words of size equal to $\log_2 S$) , the computational time (in table lookups), and the amount of data (without rekeying, this is the keystream length), respectively.

The preprocessing phase is as follows. Select M random internal initial states $\{x_i\}_{i=1}$^M. For each of the x_i, compute the output string y_i, of length logN, and store the pair (x_i ,y_i) in increasing order of y_i in RAM. In the realtime phase, given a known keystream of D+logN-1 bits, a sliding window is used to produce all D possible strings of length logN. These strings are then used to look up the precomputed table. If a match is found with y_i, then the internal state was x_i . The time-memory tradeoffs satisfy TM = S, P=M and N=T.

Time/Memory/Data tradeoff attacks are no better than exhaustive key search due to the large internal state of MUGI.

Divide and conquer attacks on keystream generators work on each component of the keystream generator separately, and sequentially solve for the subkeys. Generally, for shift register based keystream generators, the objective of the attack is to recover the initial contents of a subset of the component shift registers from a known segment of the keystream sequence. Using a divide and conquer attack on MUGI that determines the individual shift register states sequentially reduces the overall number of internal states to be searched from a product of $2^{1024} * 2^{192}$ to a sum of $2^{1024} + 2^{192}$. The approach is to guess an initial state of a component, then use some means of evaluating the guess (such as a linear consistency test or correlation attack). Fast correlation attacks reconstruct the initial state of the component without having to test exhaustively. Note that this is worse than exhaustive key search.

For MUGI, the two shift registers are not independent: each feeds into the other. This makes the divide and conquer attack outlined above difficult. It may be better to consider MUGI as a single 19 stage shift register. Now for the divide and conquer attack, consider the division of the state space into stages rather than registers. The attack should target stage a1, because it is used as input to F function in updating a1 and a2 (there are no other nonlinear operations) – see Section 2.

Correlation Attacks

Correlation between two binary segments is a measure of the extent to which they approximate each other. Correlation attacks on shift register based keystream generators are based on statistical dependencies between the observed keystream sequence and underlying shift register sequences. For divide and conquer correlation attacks, the idea is to use the correlation between the known keystream segment and the shift register sequences to sequentially recover the initial contents of the shift registers. For MUGI, a divide and conquer correlation attack requires a measure of correlation between the internal states of a and the output sequence (the contents of one third of the internal state).

This approach is compounded by the buffer b and state a not being autonomous. No measure of correlation was identified to improve this attack over exhaustive search.

An alternative to correlation for divide and conquer attacks, to determine whether a guessed value of a1 is correct, is to use linear consistency checks based upon numerous buffer equations. To date, these investigations have not led to an effective cryptanalysis.

Under the resynchronization scenario, the extra information available by having multiple keystreams encrypted under different key/IV combinations provides further opportunities for cryptanalysis. However, to date our investigations have not discovered an effective method of attack using this approach.

6 Comparison of MUGI with PANAMA and MULTI-S01

6.1 Comparison with PANAMA

In this section the evaluators discuss several structural differences between MUGI and PANAMA. The evaluators note that MUGI is stated to be designed "in the style" of Panama. Before discussing the differences the evaluators consider the similarities.

Both ciphers employ a design of two finite state machines that interact. One of these is a linear device and the other is nonlinear. The output of both ciphers is taken from the output of the nonlinear section. Both ciphers use a natural block size that is greater than a single word on the current generation of popular microprocessors, and so they output several 32-bit words for each of their internal cycles. This allows the computational cost of the cipher to be amortised over several output words, thus increasing the speed of encryption.

Panama uses blocks of 8 words each, and has a linear part with 32 of these blocks, thus the memory size of the linear part is 256 words. The nonlinear part of PANAMA has two of these blocks and a single word of feedback, making 17 words. The state size of PANAMA is 273 words.

In contrast, MUGI uses a block size of 64 bits (also called a *unit*) and has a linear part with 16 of these blocks and a nonlinear part with 3 blocks, making a state size of 38 words. This is a lot fewer than PANAMA.

The nonlinear stage of PANAMA has a special operation on 17 words that is very software efficient considering the good properties it has. The combination of bit-sliced (i.e. parallel) Boolean operations defines each output bit as a highly nonlinear function of 7 input bits, all from different input words. The internal rotations ensure good diffusion. The symmetry properties of the mapping ensure that every bit in all of its 17-words of output is equally strong. There are no weakly modified bits.

In contrast, MUGI uses one round of a target heavy Feistel structure as its nonlinear stage. One third of the state is not changed during this process. The other bits are all highly nonlinear functions of 33 bits from 3 of the 6 input words. The properties of this pair of identical F-functions were discussed earlier in Section 5.1. This nonlinear part is very different to that used in PANAMA and this difference makes it difficult to carry the security arguments for PANAMA across to MUGI. The structure of MUGI needs more analysis as it has some weaknesses that are discussed in Section 5. These properties of MUGI need further investigation.

The difference between these ciphers that should impact the most upon the security of the schemes is the exact way the feedback networks are arranged. In particular, PANAMA does not use the keystream output for any feedback, whereas MUGI uses all the output as feedback into the next nonlinear stage. This use of output must be considered a weakness compared with the prudent design choice made in PANAMA. Recall also that PANAMA has much more state memory than MUGI. These serious structural differences indicate that PANAMA may be more secure than MUGI.

6.2 Comparison of MUGI and MULTI-S01

In order to compare MUGI and MULTI-S01 the evaluators reference the Executive Summary of their Evaluation Report on MULTI-S01 from January 17 2001 [D+01]. In this report the evaluators highlighted a major flaw in the integrity check mechanism. MUGI does not have this flaw since MUGI only aims to provide a data confidentiality service not a data integrity service as well.

The confidentiality service for MUGI can best be compared to that of MULTI-S01 using a comparison of PANAMA since the actual keystream for encryption for MULTI-S01 is produced using PANAMA. For the comparison of MUGI and PANAMA refer to Section 6.1.

7 Implementation

The following section discusses some implementation issues, including the procedure MUGI uses to resynchronize, some metrics for a software implementation of MUGI, and the feasibility of a fast hardware implementation.

7.1 Rekeying Procedure

Communication systems that are based upon unreliable channels (such as IP or wireless protocols) require frequent resychronisation for synchronous stream ciphers like MUGI. This involves all entities involved in a session rekeying the algorithm at particular points in time.

The MUGI specification makes no explicit mention of rekeying. However, it provides a mechanism for inserting public Initialization Vector (IV) material into the state, and indirectly into the buffer. This occurs during stage three of the initialisation process, when the 128-bit IV is split, and its halves combined with state variables a_0 and a_1 respectively. A condensed version of the IV is combined with a_2.

The combination of the key K and IV acts as a new key without the need for expensive key agreement or out-of-band key transport protocols. However, the buffer and state contents at the end of stage two must be cached and indexed against K, or the entire initialization process must be revisited. There is a time-memory tradeoff here, with the former approach consuming 1216 bytes per key, and the latter 16 rounds of the ρ function.

[Hit01] notes that the initialisation process for the MUGI cipher makes it resistant against resynchronisation attacks. The evaluators agree with this comment.

7.2 Software Implementation

In software, a MUGI round can be implemented as two invocations of the AES-like F function, with additional operations being two fixed rotations, seven XORs and seven assignments. For all but the last stage of the initialisation phase, only one invocation of the F-function is needed per round. The seemingly expensive rotation of the sixteen buffer words can be modelled instead as a circular buffer with a decrementing offset variable, involving two additions and a mask, rather than seventeen assignments.

The F function is very simple, consisting of a single XOR of state and buffer material, followed by eight parallel applications of a (8, 8) s-box and two intertwined MDS matrix multiplications. The MDS matrices can be integrated into the s-boxes to form four (8, 32) s-boxes, each applied twice in the course of the round.

A test implementation using these techniques was developed on an Intel Pentium III 1 GHz machine with a 32 Kb L1 cache. The implementation was coded in ANSI-C and compiled with gcc on a Linux 2.4 kernel. The throughput of the cipher was 320 Megabits/second.

This result was obtained by calling the API many times with a limited set of data to avoid cache penalties. Unlike the evaluation in [Hit01], which reports a speed of 294 Megabits/second on a Pentium III 800 MHz, this result considers all the factors involved in calling the MUGI API, including setting up the stack on function invocation, tearing it down on exit, and writing the update result to memory. This is a more realistic model for measuring throughput. In either case, the positive result can be attributed to the simplicity of the round function. The throughput is expected to benefit significantly when the cipher is implemented in the architecture's native assembly language.

The size of the implementation was 1776 bytes plus an additional 4096 bytes for the s-boxes. This can be reduced by about two-thirds at the expense of approximately fifty percent of throughput.

It should be noted that because MUGI operates on 64-bit words, it will perform more efficiently on 64-bit architectures than the 32-bit architectures widely used today (such as the test platform). For example, a rotation on 64-bit words can be natively modelled as two shifts and an or operation. When implemented on a 32-bit platform, this becomes four shifts, two ors and an assignment, with a consequent drop in efficency.

7.3 Hardware Implementation

We did not develop a hardware implementation of MUGI. However, the cipher appears amenable to parallelization and ASIC implementations in the range of a few Gigabits/second should be easy to achieve. Within the F function, the MDS can be incorporated into the s-boxes, and the result of each s-box computed independently. Each state variable in the r-function can also be calculated independently. The buffer is linear and has dependencies only upon values in the previous round. The rounds of the algorithm must be executed serially.

8 References

[Bab95] Babbage, S. *A space/time tradeoff in exhaustive search attacks on stream ciphers*. European Convention on Security and Detection, IEE Conference Publication, 408, 1995.

[BS93] Biham, E., and Shamir, A. *Differential Cryptanalysis of the Full 16-Round DES*. Proceedings of Crypto'92, LNCS 740, Springer-Verlag, December 1991, pages 487-496.

[CHJ01] D. Coppersmith, S. Halevi and C. Jutla, “Cryptanalysis of stream ciphers with Linear Masking” available at the iacr eprint archive, 2002.

[D+01] Dawson, E., et. al. *MULTI-SO1 Evaluation Report*, 2001.

[DC98] Daemen, J., and Clapp, C. *Fast hashing and stream encryption with PANAMA*. Fast Software Encryption: 5th International Workshop, LNCS 1372, Springer-Verlag, 1998, pages 60-74.

[DCGNR98] Dawson E, Clark A, Gustafson H, Nielsen L, and Rutherford M. Crypt-X'98- a graphical package for the statistical testing of stream ciphers, block ciphers and key generators. 1998. QUT. QUT. Brisbane. Version 1.0.

[DR98] Daemen, J., and Rijmen, V. AES Proposal: Rijndael, available at www.esat.kuleuven.ac.be/~rijmen/rijndael/, 1998.

[Goli97] J. Dj. Golic, *``Linear Statistical Weakness of alleged RC4 keystream generator"*, Advances in Cryptology - Eurocrypt'97, LNCS vol. 1233, pp. 226-238, Springer-Verlag, 1997.

[Hit01] Hitachi Ltd. MUGI Pseudo Number Random Generator – Self-evaluation Report version 1.1, available at www.sdl.hitachi.co.jp/crypto/mugi/mugi_eve.pdf

[Mat94] Matsui, M. *Linear Cryptanalysis Method for DES Cipher*. Advances in Cryptology - EUROCRYPT '93, Springer-Verlag, 1994, pp. 386-397.

[WFHP02] Watanabe, D., Furuya, S., Yoshida, H., and Preneel, B. A New Keystream Generator MUGI. Fast Software Encryption: 9th International Workshop, LNCS 2365, Springer-Verlag, 2002, pages 179-194.

Appendix A.

The Derivation of General Buffer Change Equations Over 16 Iterations.

First consider n=0,1,2,3. We look at n=3 as the others are similar after a time shifted first line.

Let n=3 Then, from the buffer definition we have b3[t]=b4[t+1] + b7[t].

By substituting for b7 we obtain b3[t]=b4[t+1] + b10[t+3] + R32(b13[t+2]).
Substituting for b13 gives

b3[t]= b4[t+1] + b10[t+3] + R32(a0[t+4]) + R32(b3[t+8])………………….1

Now this can be shifted by t=t+8 and rotate all values by 32 giving

R32(b3[t+8])=R32(b4[t+9]) + R32(b10[t+11]) + a0[t+12] +b3[t+16]……….2

Now substitute 2 into 1 and gather b3 terms to the left hand side.

b3[t]+b3[t+16] = a0[t+12] + b4[t+1] + b10[t+3]
+ R32(a0[t+4] + b4[t+9] + b10[t+11])……………………..3

Which satisfies the general form stated earlier.
The above process can be repeated for n=0,1, and 2 by starting with

b0[t]=b4[t+4] + b7[t+3],
b1[t]=b4[t+3] + b7[t+2],
and
b2[t]=b4[t+2] + b7[t+1].

These all lead to the same general form that is parameterised by n.

Now consider n=4,5,6,7,8, and 9. We first look at n=9 as the others are similar after a time shifted first line.

Let n=9 Then, from the buffer definition we have b9[t]=b10[t+1] + R32(b13[t]).

By substituting for b13 we obtain b9[t]=b10[t+1] + R32(b0[t+3] + a0[t+2]).
Substituting for b0 gives

b9[t]= b10[t+1] + R32(b4[t+7] + b7[t+6] + a0[t+2])

we note that b7[t+6]=b9[t+8], so we have

b9[t]= b10[t+1] + R32(b4[t+7] + b9[t+8] + a0[t+2])……………………………4

Now this can be shifted by t=t+8 and rotate all values by 32 giving

R32(b9[t+8])=R32(b10[t+9]) + b4[t+15]) + b9[t+16] + a0[t+10]……..………..5

Now substitute 5 into 4 and gather b9 terms to the left hand side.

b9[t]+b9[t+16] = a0[t+10] + b4[t+15] + b10[t+1]
+ R32(a0[t+2] + b4[t+7] + b10[t+9])……………………..3

Which satisfies the general form stated earlier.
The above process can be repeated for n=4,5,6,7 and 8 by starting with

b4[t]=b10[t+6] + R32(b13[t+5]),
b5[t]= b10[t+5] + R32(b13[t+4]),
b6[t]= b10[t+4] + R32(b13[t+3]),
b7[t]= b10[t+3] + R32(b13[t+2]),
and
b8[t]=b10[t+2] + R32(b13[t+1]).

These all lead to the same general form that is parameterised by n.

Now consider n=10,11,12,13,14, and 15. We first look at n=15 as the others are similar after a time shifted first line.

Let n=15 Then, from the buffer definition we have b15[t]=b0[t+1] + a0[t].

By substituting for b0 we obtain b15[t]=b4[t+5] + b7[t+4] + a0[t].
Substituting for b7 gives and shifting b13[t+6] to b15[t+8] gives

B15[t]= b4[t+5] + b10[t+7] + R32(b15[t+8]) + a0[t]……………………….…………7

Now this can be shifted by t=t+8 and rotate all values by 32 giving

R32(b15[t+8])=R32(b4[t+13]) + b10[t+15] + a0[t+8]) + b15[t+16]……..………..8

Now substitute 8 into 7 and gather b9 terms to the left hand side.

b15[t]+b15[t+16] = a0[t] + b4[t+5] + b10[t+7]
+ R32(a0[t+8] + b4[t+13] + b10[t+15])…………………………..9

Which satisfies the general form stated earlier.
The above process can be repeated for n=10,11,12,13 and 14 by starting with

b10[t]= b0[t+6] + a0[t+5] ,
b11[t]= b0[t+5] + a0[t+4] ,
b12[t]= b0[t+4] + a0[t+3] ,
b13[t]= b0[t+3] + a0[t+2] ,
and

b14[t]= b0[t+2] + a0[t+1.

These all lead to the same general form that is parameterised by n.

Appendix B. CRYPT-X Statistical Tests

This appendix gives a mathematical description of the statistical tests used from the CRYPT-X statistical package. In each case an example is given to illustrate a particular test. The first five tests examine the hypothesis that the bit stream was based on Bernoulli trials where the proportion of ones and zeros is $\frac{1}{2}$. The two complexity tests examine the knowledge that a small subsection of the bit stream can be used to produce the remainder of the stream. If this is possible the string would not be considered to be random, especially in relation to its use in a stream cipher.
The recommended size of a sample stream to test depends on the size of the average message which is being encrypted using the keystream. i.e. if an average cryptogram has size five million bits then one should use test samples of this length. It should be noted that not all of the tests can be applied to a string of this length due to computational limitations. For example, in the linear complexity test one would need to examine a smaller substring of the keystream. It is recommended that strings of length at least 100000 bits be used for testing.

Frequency Test

The *frequency test* checks that there is an equal proportion of ones and zeros in the bit stream. For randomness the proportion of ones and zeros in the bit stream should be approximately equal, since any substantial deviation from equality could result in a successful cryptanalytic attack on the cipher. For example, assume that a cryptanalyst attacking the stream cipher knows the type of plaintext being used, e.g. standard English text coded in 8-bit ASCII, and the keystream has $\frac{3}{4}$ of the bits zero. Under this assumption the cryptanalyst knows the frequency distribution of the plaintext in terms of single bits, digraphs and trigraphs. With this knowledge the cryptanalyst could recover a substantial amount of the plaintext, using ciphertext alone.
The number of ones in a random binary sequence follows a binomial distribution, with mean $\frac{n}{2}$ and variance $\frac{n}{4}$. This may be approximated using a normal distribution. The following notation is used:

$n = \text{total number of bits};$

$n_0 = \text{number of zeros};$

$n_1 = \text{number of ones};$

$\hat{p} = \frac{n_1}{n} = \text{proportion of ones in the sequence}.$

The aim of the frequency test is to determine how the proportion of ones, $\hat{p}$, in the sample stream of length n bits, fits into the hypothesised distribution where the proportion of ones, $\pi = 0.5$ and the variance, $\sigma^2 = \frac{1}{4n}$. This is a two-tailed test [BHAT 77]. The standardised normal test statistic is: $z = 2\sqrt{n}(\hat{p} - 0.5)$. The significance probability value, p, of the normal distribution is calculated for this statistic. This measures the probability of obtaining a number of ones equal to or further from the mean of $\frac{n}{2}$ than this sample gives for the hypothesised (where $\pi = 0.5$ and $\sigma^2 = \frac{1}{4n}$).
A small significance probability indicates a significant result (i.e., the stream is considered to be non-random). For large values of n ($n > 100000$) a highly significant result (significance probability < 0.001) indicates a possible weakness in the cipher and it is recommended that no further tests be carried out on this sample as the imbalance of ones and zeros may effect their results.

It should be noted that passing the frequency test does not mean the stream is not patterned. The following highly patterned streams, where the number of ones and zeros are equal, will pass the frequency test:
11111111..........00000000.........
10101010101010....................
Hence further testing is required to obtain knowledge of any patterns in the stream.
Example:
Test stream:
10100010000101110001011000111010101010101010000001
Calculations and results:

$$n = 50$$

$$\sigma^2 = \tfrac{1}{4\times 50}$$

$$n_1 = 21$$

$$\hat{p} = 0.42$$

$$z = \sqrt{50}(0.42 - 0.5) = -1.13137$$

$$p = 0.2579$$

Interpretation:

25.79 % of bit streams of length 50 will have a number of ones equal to or further from the mean of 25, for the hypothesised distribution, than this sample. This sample satisfies the frequency test.

Binary Derivative Test

The binary derivative is a new stream formed by the exclusive-or operation on successive bits in the stream. Successive binary derivative streams may be obtained from each new binary derivative, each one being of length one less than its predecessor [CARR 88].
The proportion of ones in the *i*-th binary derivative gives the proportion of overlapping (*i*+1)-tuples from the original stream in one of two known groupings of these (*i*+1)-tuples. This will be explained for $i = 1$ and $i = 2$.
When $i = 1$ (first binary derivative) we are looking at the overlapping two-tuples: 00, 01, 10, 11 (in the original stream).
The proportion of ones in the first binary derivative, $\hat{p}(1)$, gives the proportion of the total number of 01 and 10 patterns in the original stream.
$\hat{p}(1) >$ ½ means there is a larger proportion of the group of 01 and 10 two-tuples (in the original stream).
$\hat{p}(1) <$ ½ means there is a larger proportion of the group of 00 and 11 two-tuples (in the original stream).
A combination of the frequency test on the original stream and its first binary derivative is equivalent to testing that there is an equal number of these four overlapping two-tuples in the original stream. This replaces the well-known Serial Test [DAWS 91].
When $i = 2$ (second binary derivative) we are looking at overlapping three-tuples: 000, 001, 010, 011, 100, 101, 110, 111 (in the original stream). The proportion of ones in the second binary derivative, $\hat{p}(2)$, gives the proportion of the total number of 001, 011, 100, 110 patterns in the original stream.
$\hat{p}(2)$ > ½ means there is a larger proportion of the group of 001, 100, 110, and 011 three-tuples.

$\hat{p}(2)$ < ½ means there is a larger proportion of the group of 000, 010, 101, and 111 three-tuples.
A combination of the frequency test on the original stream and a similar test on the first and second binary derivatives, tests that there is an equal number of the eight overlapping three-tuples in the original stream, for practically all cases. If a cipher gives a satisfactory result to these tests AND also the change point test, then it can be considered to generate equal numbers of the overlapping three-tuples.
Notation:

$$n_1(i) = \text{number of ones in the } i\text{-th derivative}$$

$$\hat{p}(i) = \frac{n_1(i)}{n-i} = \text{proportion of ones in the } i\text{-th derivative}$$

The frequency test is applied to each stream and the standardised normal variable is found for the proportion of ones in each of the first two binary derivatives: $z(i) = 2\sqrt{n-i}(\hat{p}(i) - 0.5)$, for $i = 1,2$.
The significance probability value, p_i, of the normal distribution is calculated for each statistic. A small significance probability indicates a significant result. For large n ($n > 100000$) a highly significant result (significance probability < 0.001) indicates a possible weakness in the cipher.
Example:
Test stream:

10100010000101110001011000111010101010101010000001

Calculations and results:

D_1 : 1110011000111001001110100100111111111111111000001
D_2 : 001010100100101101001110110100000000000000100001

Frequency test on first binary derivative (D_1) :

$$\mathrm{n}_1(1) = 30$$
$$\tfrac{\mathrm{n}-1}{2} = 24.5$$
$$\hat{p}(1) = \tfrac{n_1(1)}{n-1} = \tfrac{30}{50-1} = 0.61224$$
$$\mathrm{z}(1) = 2\sqrt{49}(0.61224 - 0.5) = 1.57143$$
$$p_1 = 0.1161$$

Interpretation:

11.61 % of bit streams of length 49 will have a number of ones equal to or further from the mean of 24.5, for the hypothesised distribution, than this sample. This sample satisfies the frequency test on the first binary derivative.

Since the frequency test is satisfied for the original stream and the first binary derivative then the cipher can be regarded as producing an equal number of overlapping two-tuples.
Frequency test on second binary derivative (D_2) :

$$\mathrm{n}_1(2) = 16$$
$$\tfrac{\mathrm{n}-2}{2} = 24$$
$$\hat{p}(2) = \tfrac{n_1(2)}{n-2} = \tfrac{16}{50-2} = 0.333$$
$$\mathrm{z}(2) = 2\sqrt{48}(0.333 - 0.5) = -2.3094$$
$$p_2 = 0.0209$$

Interpretation:

> 2.09 % of bit streams of length 48 will have a number of ones equal to or further from the mean of 24, for the hypothesised distribution, than this sample. This sample satisfies the frequency test on the second binary derivative.

Even though the frequency tests on the original stream and the first and second binary derivatives were all satisfied, the cipher will still have to satisfy the change point test before regarding it as producing an equal number of overlapping three-tuples.

Change Point Test

At each bit position, t, in the stream the proportion of ones to that point is compared to the proportion of ones in the remaining stream.
The difference or *change* in these proportions is compared for all positions in the bit stream. The bit where the maximum change occurs is called the *change point*. The test applied determines whether this *change* is significant for a binomial distribution where the proportion of ones in the stream is expected to be 0.5.
This test is very useful for detecting patterned streams which have passed the frequency test on the stream and the first two binary derivatives. Even if $\pi = \frac{1}{2}$ and the stream has passed the frequency test it could be, for n = 10^6, that $\pi = \frac{1}{4}$ for the first 500000 bits and $\pi = \frac{3}{4}$ for the second 500000 bits. This is not considered to be a good pseudorandom sequence to be used as a keystream, and the change point test would detect such cases.
This test is also useful for checking that there is an equal number of overlapping three-tuples for streams which have passed the frequency test on the original stream and also on the first two binary derivatives.
The hypothesis to be tested is that there is no change in the proportion of ones throughout the whole stream. The statistic [PETT 79] used is $\mathrm{U[t] = n \times S[t] - t \times S[n]}$ where

> $n = \text{total bits in stream}$
>
> $S[n] = \text{total ones in stream}$
>
> $S[t] = \text{number of ones to bit } t$

The maximum absolute value of this statistic is found:

> $\text{Max} = \text{Maximum of ABS}(U[t]), \text{for } t = 1 \ldots n$

The significance probability, *p*, associated with this statistic is approximated by:

$$p = e^{-\frac{2Max^2}{nS[n](n-S[n])}}.$$

For small values of p the actual significance probability is smaller than that calculated. The smaller the value of p then the more significant the result. For large streams a highly significant result, p < 0.001, indicates a possible weakness in the algorithm.

Example:

Test stream:

s = 10100010000101110001011000111010101010101010000001

Calculations and results:

$n = 50$

$S[n] = 21$

$t = 43$

$S[t] = 20$

$\text{Max} = |50 \times 20 - 43 \times 21| = 97$

$p = e^{-\frac{2 \times 97^2}{50 \times 21(50-21)}} = 0.5390$

Interpretation:

The actual significance probability of the change in the proportion of ones is less than 53.9%. This result indicates there is no significant change in the proportion of ones in the bit stream. This sample satisfies the change point test.

Subblock Test

The stream is divided into S non-overlapping subblocks, each of length b. Any fractional subblocks remaining are ignored. For a stream of length *n*, the number of subblocks is the integral part of $\lfloor \frac{n}{b} \rfloor$, i.e. S = $\lfloor \frac{n}{b} \rfloor$.

For a subblock size of b ≤ 16 a test of uniformity is applied – i.e., there should be an equal number of each b bit pattern. The test compares the observed number of each b bit pattern with $S/2^b$.

The test statistic used is $\chi^2 = \frac{2^b}{S} \sum_{i=0}^{2^b-1} f_i^{\,2} - S$ [BEKE 82], where f_i is the frequency of subblock pattern whose equivalent decimal value is i. This statistic is compared with a chi-square distribution with degrees of freedom equal to $2^b - 1$. For values of b > 6 the normal distribution may be used to approximate the chi-square distribution. Limitations: The minimum length required for the stream to test for randomness using b-bit subblocks is $5b \times 2^b$ bits.

For a subblock size of b > 16 the repetition test [GUST 96], is applied. The repetition test measures the number of repeated patterns in a sample of S subblocks, each containing b bits. Given the binary stream is divided into S b-bit subblocks then, for a random stream, each of the $N = 2^b$ possible binary b-bit patterns is equally likely to occur. As the block length increases and $N \to \infty$, with a sample of size $S \to \infty$ where $\frac{S}{N} \to 0$, then the distribution of the number of subblock repetitions in the sample approaches a Poisson distribution with a mean of $\lambda = S - N(1 - e^{-\frac{S}{N}})$. When $S = 8\sqrt{N}$ the mean converges to 32, for large values of b (say b > 16). The Poisson distribution is well approximated by the normal distribution for $\lambda = 32$.

The test requires a count of the number of subblock repetitions, r. (Note that if a particular pattern occurs three times, then this would add two to the number of repetitions).

The number of b-bit subblocks required for the test is $S = 8\sqrt{N}$, and gives $\lambda \approx 32$.

The procedure is to sort the subblocks and then determine the number of repetitions, *r*.

The test statistic is $z = \frac{r - \lambda}{\sqrt{\lambda}}$ (standard normal statistic for a Poisson distribution with a mean equal to λ), and is compared with the standard normal distribution. A two-tailed test applies since both too few or too many repetitions may indicate non-randomness of the stream.
The required stream length to apply the repetition test using b-bit subblocks is $b \times 2^{\frac{b}{2}+3}$ bits. This is considerably less than the length of stream required to apply the uniformity test for subblocks of the same size. Since the stream lengths required are very large, no sample stream will be shown. Instead, the following data will be used to illustrate a test calculation for the uniformity test:

$b = 8$ (hence the uniformity test is applied)

$n = 100000$

$S = \left\lfloor \frac{100000}{8} \right\rfloor = 12500$

Number of 8 bit patterns $= 2^8 = 256$

Assume $f_0 = 45, f_1 = 50, \ldots, f_{255} = 44$, to give:

$\chi^2 = \frac{2^8}{12500} \sum_{i=0}^{255} f_i^2 - 12500 = 260$ (say)

Degrees of freedom = 255

$z = \sqrt{2 \times 260} - \sqrt{2 \times 255 - 1} = 0.24248$

$p = 0.4042$

Interpretation:

40.42% of all possible streams of length 100000 will have a distribution of 8-bit subblocks less uniform than this sample shows. This sample satisfies the subblock test for subblocks of length 8.

The following data is used to illustrate a test calculation for the repetition test:

$b = 18$ (hence the repetition test is applied)

$n = 100000$

$N = 2^{18} = 262144$

$S = 2^{\frac{18}{2}+3} = 4096$

Stream length tested $= 4096 \times 18 = 36864$ bits

$r = 38$ (say)

$z = \frac{38 - 32}{\sqrt{32}} = 1.06066$

$p = 0.1444$

Interpretation:

14.44% of all possible streams of length 36864 will have a 18-bit subblock repetition count further from the mean (32) than this sample shows. This sample satisfies the subblock test for subblocks of length 18.

Runs Test

The runs distribution test compares the distribution of the number of runs of ones (blocks) and zeros (gaps) with that expected under randomness. For a random binary stream where $\Pr(1) = \Pr(0) = \frac{1}{2}$ there should be an equal number of number of blocks and gaps of the same length. Based on Golomb's postulates, the expected number of runs of length i for a random binary stream should be $\frac{1}{2^i}$ of the number of runs, and for each length there should be an equal number of runs of ones and zeros, i.e. $E(r_{0i}) = E(r_{1i}) = \frac{Runs}{2^{i+1}}$, where *Runs* indicates the number of runs in the binary stream. The hypothesis to be tested is that the distribution of runs in the stream fits a binomial population for which $\Pr(1) = \Pr(0) = \frac{1}{2}$. The test applied is adapted from [MOOD40]. The long runs are added together to form new variables s_{0k} and s_{1k} corresponding to the number of gaps and blocks of length k or more, where $s_{0k} = \sum_{i=k}^{n_0} r_{0i}$ and n_0 is the number of zeros in the stream.

By adding the long runs together a certain amount of information will be lost. In order to minimise the amount of information lost, it is recommended here that $k = \left\lfloor \log_2 \frac{n+1}{5} - 1 \right\rfloor$.

For a stream of length $n = 10^6$ this would give a maximum value of k = 16, and hence the number of gaps of length 16 or more would be added together to give $s_{0,16}$ and the number of blocks of length 16 or more would be added together to give $s_{1,16}$.

Explanation of terms:

n = number of bits in stream

n_1 = number of ones in the bit stream

r_{0i} = number of runs of 0 of length i

s_{0i} = number of runs of 0 of length i for $i < k$

s_{0k} = number of runs of 0 for lengths $\geq k$

r_{1i} = number of runs of 1 of length i

s_{1i} = number of runs of 1 of length i for $i < k$

s_{1k} = number of runs of 1 for lengths $\geq k$

The variables:

$$u_i = \frac{r_{1i} - n(\frac{1}{2})^{2+i}}{\sqrt{n}} \quad i = 1,\dots,k-1$$

$$u_k = x_k = \frac{s_{1k} - n(\frac{1}{2})^{k+1}}{\sqrt{n}}$$

$$u_{k+i} = y_i = \frac{r_{0i} - n(\frac{1}{2})^{2+i}}{\sqrt{n}} \quad i = 1,\dots,k-1$$

$$u_{2k} = z = \frac{n_1 - \frac{1}{2}n}{\sqrt{n}}$$

are asymptotically normally distributed with zero means and variances and covariances:

$$\sigma(x_i, x_i) = \sigma(y_i, y_i) = (\tfrac{1}{2})^{i+2} - (2i-1)(\tfrac{1}{2})^{2i+4}$$

$$\sigma(x_i, x_j) = \sigma(y_i, y_j) = (1-i-j)(\tfrac{1}{2})^{i+j+4}$$

$$\sigma(x_i, x_k) = -(i+k)(\tfrac{1}{2})^{i+k+3}$$
$$\sigma(x_k, x_k) = (\tfrac{1}{2})^{k+1} - (2k+1)(\tfrac{1}{2})^{2k+2}$$
$$\sigma(x_i, y_j) = (5-i-j)(\tfrac{1}{2})^{i+j+4}$$
$$\sigma(x_k, y_j) = (4-k-j)(\tfrac{1}{2})^{k+j+3}$$
$$\sigma(x_i, z) = \sigma(y_i, z) = (i-2)(\tfrac{1}{2})^{i+3}$$
$$\sigma(x_k, z) = (k-5)(\tfrac{1}{2})^{k+2}$$
$$\sigma(z,z) = \sigma_{zz} = \tfrac{1}{4}$$

Test procedure:

1. Determine k.
2. Take a sample stream of n bits from a stream cipher. Determine the number of runs of each length to give s_{1i} and s_{0i} for $i = 1,\ldots,k$.
3. Calculate u_j for $j = 1,\ldots,2k$ using above formulae.
3. Determine $S = [\sigma_{ij}]$ which is a $2k \times 2k$ matrix. Calculate $S^{-1} = [\sigma^{ij}] = [\sigma_{ij}]^{-1}$.

   This will require obtaining the inverse of a matrix of up to 32^2 (1024) elements for $n \le 10^6$ bits. Calculate $Q = \mathbf{u}^T S^{-1} \mathbf{u} = \sum \sigma^{ij} u_i u_j$ which follows a χ^2_{2k} distribution (chi-squared distribution with 2k degrees of freedom). There are $(2k)^2$ terms in this sum.

$$Q = \sum_{i=1}^{2k} \sigma^{ii} {u_i}^2 + 2\sum_{i<j} \sigma^{ij} u_i u_j$$

The significance probability value, p, of the chi-squared distribution is calculated for this statistic. A small value of p indicates a significant result. For large streams a highly significant result, $p < 0.1\%$, indicates a possible weakness in the algorithm.

The runs test can be used to support results from the previous tests. Failure of the runs test indicates that there is a bad distribution of run lengths or that there are no runs recorded above a certain length that are expected to occur for streams of the sample size. The zero frequencies recorded will result in a higher chi-squared statistic thus giving a smaller significance probability.

Example:

Test stream:

10100010000101110001011000111010101010101010000001

Calculations and results:

$$n = 50$$
$$\text{total runs} = 31$$
$$n_1 = 21$$
$$k = \lfloor \log_2 \tfrac{n+1}{5} - 1 \rfloor = \lfloor \log_2 \tfrac{51}{5} - 1 \rfloor = 2$$
$$s_{01} = 10, s_{02} = 5, s_{11} = 13, s_{12} = 3$$
$$u_1 = x_1 = \frac{13 - 50(\tfrac{1}{2})^3}{\sqrt{50}} = 0.9545941546018$$
$$u_2 = x_2 = \frac{3 - 50(\tfrac{1}{2})^3}{\sqrt{50}} = -0.4596194077713$$

$$u_3 = y_1 = \frac{10 - 50(\frac{1}{2})^3}{\sqrt{50}} = 0.5303300858899$$

$$u_4 = z = \frac{21 - \frac{1}{2}(50)}{\sqrt{50}} = -0.5656854249492$$

$$\sigma(u_1,u_1) = \sigma(u_3,u_3) = (\tfrac{1}{2})^3 - (2-1)(\tfrac{1}{2})^6 = 0.109375$$

$$\sigma(u_1,u_2) = -3(\tfrac{1}{2})^6 = -0.046875$$

$$\sigma(u_2,u_2) = (\tfrac{1}{2})^3 - 5(\tfrac{1}{2})^6 = 0.046875$$

$$\sigma(u_1,u_3) = 3(\tfrac{1}{2})^6 = 0.046875$$

$$\sigma(u_2,u_3) = 1(\tfrac{1}{2})^6 = 0.015625$$

$$\sigma(u_1,u_4) = \sigma(u_3,u_4) = -1(\tfrac{1}{2})^4 = -0.0625$$

$$\sigma(u_2,u_4) = -3(\tfrac{1}{2})^4 = -0.1875$$

$$\sigma(u_4,u_4) = \sigma_{44} = 0.25$$

Elements of the inverse matrix, S^{-1} :

$\sigma^{11} = 6\frac{2}{3}, \sigma^{12} = -5\frac{1}{3}, \sigma^{13} = -4, \sigma^{14} = -3\frac{1}{3}, \sigma^{21} = -5\frac{1}{3}, \sigma^{22} = -5\frac{1}{3}, \sigma^{23} = 0, \sigma^{24} = -5\frac{1}{3},$

$\sigma^{31} = -4, \sigma^{32} = 0, \sigma^{33} = 12, \sigma^{34} = 2, \sigma^{41} = -3\frac{1}{3}, \sigma^{42} = -5\frac{1}{3}, \sigma^{43} = 2, \sigma^{44} = -\frac{1}{3}.$

Q = 8.4733.. follows a χ^2_4 distribution.

p = 0.076

Interpretation:

7.6% of bit streams of length 50 will have a distribution of run lengths further from the expected distribution than this sample gives. This sample satisfies the runs distribution test.

The length of the longest run was also recorded.

Given a bit stream of length N, the expected number of runs = $\mathbf{\frac{N+1}{2}}$.

Hence, for a bit stream of length 2^n, the expected number of runs ≈ 2^{n-1}.

Applying Golomb's Postulates, it is expected that $\frac{1}{2^i}$ of the runs have length i in an infinite random binary stream. So in a random bit stream of length 2^n, the expected number of runs of length k ≈ 2^{n-k-1}.

Sequence Complexity Test

The sequence complexity, c(s), is the number of different substrings encountered as the stream, s, is viewed from beginning to end [LEMP 76].

Example (n = 16) :

s = 1 0 0 1 1 1 1 0 1 1 0 0 0 0 1 0

Marking in different substrings :

s = 1/0/0 1/1 1 1 0/1 1 0 0/0 0 1 0/

Here the sequence complexity c(s) = 6

A *threshold value* of sequence complexity is used to measure the randomness of a sequence. This *threshold value* is $\frac{n}{\log_2 n}$ where n is the total bits in the stream. A stream with a sequence complexity measure below this *threshold value* would be considered to be patterned, ie not random. For the example given, the *threshold value* $= \frac{16}{4} = 4$. Hence the stream is not considered patterned.

An expected value for the sequence complexity of a random stream of the same length is calculated using the following algorithm [GUST 96]:

```
$i = 2;$
$c = 2;$
while $(i < n)$ do
begin
    $i = i + \left\lfloor \frac{\log(i-1)}{\log(2)} \right\rfloor + 2;$
    $c = c + 1;$
end;
if $(n < i)$ then $c = c - 1;$
```

It is expected that a good pseudo-random number sequence has a sequence complexity which is close to this value. It should be noted that the expected value of sequence complexity is always greater than the threshold value. However, a bit stream will only be considered to not satisfy the sequence complexity test if the value of c(s) is less than the threshold value.
The sequence complexity is used to replace the autocorrelation test which is commonly used to determine any periodicity in the pseudorandom number generator. Periodicity would greatly reduce the number of "different" substrings encountered. Hence c(s) would be low and fall below the threshold value. [DAWS 91]

Example

Test stream:

10100010000101110001011000111010101010101010000001

Calculations and results:

$\mathrm{n} = 50$

$\mathrm{c(s)} = 10$

$\text{Threshold value} = 8.859191$

$\text{Expected value} = 13$

Interpretation:

This sample stream is considered random based on the sequence complexity test.

Linear Complexity Test

(1) Linear Complexity

The linear complexity test checks for the minimum amount of knowledge (bits) needed to reconstruct the whole stream. Every finite stream, s, can be produced by a linear feedback shift register (LFSR). The length of the shortest LFSR which will produce the stream is said to be the linear complexity of the stream, which will be denoted by L(s).
If the value of L(s) is L then 2L consecutive terms can be used to reconstruct the whole sequence using the Berlekamp Massey algorithm. [MASS 69] Hence, in order to avoid stream reconstruction, the value of L should be large.

Example:

0101100101010010011110000011011100110001110101111101101

This shortest recurrence relation which will create this sequence is:

$u(t+6) = u(t+5) \oplus u(t+4) \oplus u(t+1) \oplus u(t)$

where $\oplus$ is addition mod 2, and the first bit is $u(0)$.
For example:

If $t = 0$ then $\begin{array}{c} u(6) = u(5) \oplus u(4) \oplus u(1) \oplus u(0) \\ 0 = 0 \oplus 1 \oplus 1 \oplus 0 \end{array}$.

If $t = 1$ then $\begin{array}{c} u(7) = u(6) \oplus u(5) \oplus u(2) \oplus u(1) \\ 1 = 0 \oplus 0 \oplus 0 \oplus 1 \end{array}$.

If $t = 2$ then $\begin{array}{c} u(8) = u(7) \oplus u(6) \oplus u(3) \oplus u(2) \\ 0 = 1 \oplus 0 \oplus 1 \oplus 0 \end{array}$.

This means that the linear complexity, L(s), of this sequence is six. If any twelve consecutive bits are known then the whole sequence can be reconstructed. [MASS 69]
It should be noted that some keystreams can pass all the previous tests yet possess a very small linear complexity. An example of this is an m-sequence (see [RUEP 84]). An m-sequence has a period of length $2^L - 1$ and a linear complexity of L. An m-sequence has the *best* possible distribution of zeros and ones for a sequence of period $2^L - 1$. In this fashion an m-sequence appears to be *statistically* random in terms of tests 0 to 0. In fact m-sequences are commonly used as *white noise* generators. However, in terms of their use in a stream cipher an m-sequence offers very low security. Knowledge of only 2L consecutive bits of the keystream is needed to derive the defining LFSR and hence determine the whole keystream.
For large n, L(s) is approximately normally distributed with $\mu = \frac{n}{2}, \sigma^2 = \frac{86}{81}$ [RUEP 84], [KREY 81]. Using the standardised normal statistic $z = \sqrt{\frac{81}{86}}(L(s) - \frac{n}{2})$ the significance probability value, *p*, of the normal distribution is calculated.
Since only low values of L(s) signify a possible weakness to the cipher, only a one-tailed test (lower tail) need apply. A small value of p indicates a significant result. For large streams a highly significant result ($p < 0.1\%$) indicates a possible weakness in the algorithm.
The linear complexity test by itself can classify as random, streams which may be highly patterned, or contain large substrings which are highly patterned. Some of the previous test results should support this. e.g. a stream of $\frac{n}{2} - 1$ zeros followed by a one, and then followed by a repetition of these $\frac{n}{2}$ terms, has a linear complexity of $\frac{n}{2}$. This stream would be classified as being random using the linear complexity test. Clearly, such a stream is highly patterned and would not satisfy the previous tests. However, it is possible to construct a stream of length n which would pass all the previous statistical tests, and have a linear complexity of approximately $\frac{n}{2}$ yet would contain a large highly patterned substring. Hence the following linear complexity profile tests are carried out.

(2) Linear Complexity Profile

Since some highly patterned streams can give a linear complexity measure close to $\frac{n}{2}$ a second test measures the change in the linear complexity profile of the stream as each bit is added. Let s(i) be the substring formed by taking the first i bits of s. If L(s(i)) for i = 1,...,n denotes the linear complexity of s(i) then the values of s(i) are defined to be the linear complexity profile of s and should follow approximately the $\frac{i}{2}$ line [MASS 69]. A failure in this test would highlight any large deviations from the $\frac{i}{2}$ line, which would appear for strings passing the linear complexity test and containing any large highly patterned substrings. A change in linear complexity signifies a *jump*.
There are two tests relating to the Linear Complexity Profile:

(3) Linear Complexity Profile – Number of Jumps

Let the total number of jumps be F. For large n, F is approximately normally distributed with $\mu = \frac{n}{4}$ and $\sigma^2 = \frac{n}{8}$ [CART 87]. The standardised statistic for the number of jumps is $z = \sqrt{\frac{8}{n}}(F - \frac{n}{4})$. The significance probability, *p*, for this standardised statistic is calculated. Since a small number of jumps would indicate a sequence within which patterns may exist, a one-tailed test (lower tail) is applied. A small value of p ($p < 0.1\%$) indicates that the number of jumps in linear complexity is low, and there may be patterns in the stream which would indicate a possible weakness in the cipher.

(4) Linear Complexity Profile – Jump Size

If a stream passes the test on the number of jumps in linear complexity, then the distribution of jump heights may be investigated. The height of a jump is the difference in linear complexity when a change occurs. Let the total number of *jumps* in linear complexity be F, where f_i is the number of jumps of *height* i. For a random string based on Bernoulli trials where the probability of a one on each trial is one half, the probability, p_i that a given jump has height i is given by $p_i = (\frac{1}{2})^i$. Hence the expected number of jumps of height i, e_i , is given by $e_i = p_i \times F$.

The chi-squared statistic used is $\chi^2 = \sum_{i=1}^{m} \frac{(f_i - e_i)^2}{e_i}$ [CART 87]. The maximum value of $i = m$ is determined from the condition for the chi-squared test, that $e_i > 5$. The number of degrees of freedom, $m - 1$, is determined from the sample taken.
The significance probability value, p, of the chi-squared distribution is calculated for this statistic. A small significance probability indicates a significant result – i.e., the stream is considered to be non-random. For large samples a highly significant result, $p < 0.1\%$, indicates a possible weakness in the algorithm.

Example

Test stream:

10100010000101110001011000111010101010101010000001

Calculations and results:

Linear Complexity Test

$n = 50$

$\mu = \frac{n}{2}$

$\sigma^2 = \frac{86}{81}$

$L(s) = 25$

$z = \sqrt{\frac{81}{86}}(25 - \frac{50}{2}) = 0$

$p = 0.5$

Interpretation:

50 % of bit streams of length 50 will have a linear complexity less than this sample. This sample satisfies the linear complexity test.

Hence $2 \times L(s) = 50$ bits (the whole stream) is needed to reconstruct the stream using the Berlekamp-Massey algorithm.

Linear Complexity Profile - Number of jumps

$F = 15$

$z = \sqrt{\tfrac{8}{50}}(15 - \tfrac{50}{4}) = 1$

$p = 0.8413$

Interpretation:

84.13% of streams of length 50 will have a number of jumps in linear complexity less than this sample. This sample satisfies the test on the number of jumps in linear complexity.

Linear Complexity Profile – Jumps size

$f_1 = 11,\ f_2 = 2,\ f_3 = 0,\ f_4 = 1,\ f_5 = 1.$

$\mathrm{e}_1 = 7.5$ Since $e_i < 5$ for $i > 2$, then these values are combined to give $e_{2+} = 7.5$. The corresponding values of f_i are combined to give $\mathrm{f}_{2+} = 4$.

$\chi^2 = \tfrac{(11-7.5)^2}{7.5} + \tfrac{(4-7.5)^2}{7.5} = 3.27$

$\text{Degrees of freedom} = m - 1 = 2 - 1 = 1$

$p = 0.0707$

Interpretation:

Approximately 7.07% of bit streams of length 50 will have a sump size distribution further from the expected distribution than this sample gives. The sample satisfies the test on the distribution of the linear complexity jump size.

Results of CRYPT-X Tests

The p-value obtained from a test represents the probability of obtaining a result further than the test statistic lies from that expected, if the algorithm produces a random stream. Very small p-values would support non-randomness for the given measure.

Results for Keystreams with Variable Key and Fixed IV

Length of each MUGI keystream = 2^{20} bytes (8,388,608 bits).
Length of each MUGI keystream (Complexity tests) = 100,000 bits.
Number of keystreams for each test = 100

The table below gives the number of p-values falling below 0.1, 0.05 and 0.01 from the 100 MUGI keystreams tested. In a sample of 100 keystreams, the expected count for the number of p-values less than 0.1 is 10, for 0.05 is 5, and for 0.01 is 1.

	Number of p-values below:		
Test	***.10***	***.05***	***.01***
Frequency	13	5	0
Binary Derivative (1)	12	4	0
Binary Derivative (2)	10	2	0
Change Point	24	11	2
Subblock b = 4	11	6	1
Subblock b = 8	12	7	2
Runs Distribution	3	0	0
Linear Complexity	5	5	2
LC profile - Jumps	15	7	3
LC Profile - Jump Size	21	14	5
Longest Run	Max = 34	Next = 30	
Sequence Complexity	Max = 6144	Min = 6097	

Further analysis was applied to the results when more than k of the p-values were greater than k(.01)%. An upper 99% limit for the count when k = 10 is calculated using:

$$100\left(0.1+2.326\sqrt{\frac{.1(1-.1)}{100}}\right) = 17$$

The corresponding values for k = 5 and k = 1 are: 10 and 3.
Counts falling above these values would be classified as significant.

The results for the Change Point test and the Linear Complexity Jumps Size test are the only ones that show this significance.

Longest Run

For a random bit stream of length 2^{20}, the expected number of runs of length 34 $\approx 2^{20 - 34 - 1} = 2^{-15}$. This implies that it is highly unlikely that a run of length 34 will appear in a bit stream of this length.

The expected number of runs of length 22 $\approx 2^{20 - 22 - 1} = 2^{-3} \approx 0.125$, which supports that this is a more likely occurrence for the length of the longest run.

It would appear that the length of the longest run exceeds what would be expected in bit streams of this length. It should be noted that the length of the longest run may not exceed 34 for much longer streams. Hence we cannot conclude that this result shows any weakness in the MUGI algorithm.

Sequence Complexity

For streams of length 10^5 bits the threshold value = 6,021 and the mean value = 6,056 Both the minimum and maximum values obtained are above these values, and hence there is no indication on non-randomness based on the sequence complexity test.

Linear Complexity

The results for the linear complexity profile indicate that, as the bit-stream increases in length, the changes in linear complexity maintain the expected value of half the stream length. These results support the randomness of the keystream output from MUGI, based on linear complexity, such that the whole bit-stream is required to re-construct the stream itself, thus giving an attacker no advantage in being able to create the bit-stream with a smaller number of output bits.

Bibliography

[BEKE 82] H. Beker and F.Piper, **Cipher Systems: The Protection of Communications**, Northwood Books, London, 1982.

[BHAT 77] G. Bhattacharyya and R. Johnson, **Statistical Concepts and Methods**, John Wiley & Sons, 1977.

[CARR 88] J.M. Carroll and L.E. Robbins, “Using binary derivatives to test an enhancement of DES”, **Cryptologia**, Vo1 12 number 4, 1988, pp 193-208.

[CART 87] G. Carter, “A statistical test for randomness based on the linear complexity profile of a binary sequence”, **Technical Report** for Racal Comsec Ltd., 1987.

[DAWS 91] E.P. Dawson, **Design and Cryptanalysis of Symmetric Ciphers**, PhD Thesis, Queensland University of Technology, 1991.

[GUST 96] H.M. Gustafson, **Statistical Analysis of Symmetric Ciphers**, PhD Thesis, Queensland University of Technology, Brisbane Australia, 1996.

[KREY 81] E. Kreysig, **Introductory Mathematical Statistics**, John Wiley and Sons, 1981.

[LEMP 76] A. Lempel and J. Ziv, “On the complexity of finite sequences”, **IEEE Trans. on Information Theory**, Vol.IT-22, Jan.1976,pp 75-81.

[MASS 69] J.L. Massey, “Shift register sequences and BCH decoding”, **IEEE Transactions on Information Theory**, Vol. IT-15, Jan. 1969, pp 122-127.

[MOOD 40] A.M. Mood, “The distribution theory of runs”, **Ann. Math. Statist.**, Vol 11, 1940, pp 367-392.

[PETT 79] A.N. Pettitt, “A non-parametric approach to the change - point problem”, **Appl. Statist.**, Vol. 28 No. 2, 1979, pp 126-135.

[RUEP1 84] R.A. Rueppel, **New Approaches to Stream Ciphers**, PhD Thesis, Swiss Federal Institute of Technology, 1984.

[RUEP2 84] R.A. Reuppel, “**Analysis and Design of Stream Ciphers**”, Springer-Verlag, 1986.

Appendix C. Crypt-X Results

Results of Frequency Test Applied to Bit Positions in 64-bit Subblocks

MUGI Keystreams with Variable Key and Fixed IV

Length of each MUGI keystream = 2^{20} bytes (8,388,608 bits).
Number of 64-bit subblocks = $2^{23}/2^{6} = 2^{17}$ bits (131,072 bits)
Number of keystreams for each test = 100

The table below gives the number of p-values falling below 0.1, 0.05 and 0.01 from the 100 MUGI keystreams tested. In a sample of 100 keystreams, the expected count for the number of p-values less than 0.1 is 10, for 0.05 is 5, and for 0.01 is 1.

Bit Position in 64-bit Subblocks	***Number of p-values less than:***		
	0.1	***0.05***	***0.01***
Position = 1	9	3	1
Position = 2	15	6	2
Position = 3	5	1	0
Position = 4	6	4	2
Position = 5	12	6	1
Position = 6	11	6	1
Position = 7	6	5	1
Position = 8	11	3	0
Position = 9	17	8	1
Position = 10	9	7	1
Position = 11	12	6	0
Position = 12	9	9	2
Position = 13	8	5	2
Position = 14	11	5	0
Position = 15	8	2	0
Position = 16	10	6	0
Position = 17	14	9	1
Position = 18	6	2	0
Position = 19	12	8	3
Position = 20	7	3	0
Position = 21	10	6	0
Position = 22	6	2	0
Position = 23	9	4	2
Position = 24	14	8	2
Position = 25	13	5	1
Position = 26	8	6	0
Position = 27	9	4	2
Position = 28	8	2	0
Position = 29	9	6	2
Position = 30	14	7	1
Position = 31	9	4	1
Position = 32	11	4	0
Position = 33	8	7	2

Position = 34	10	4	1
Position = 35	4	1	1
Position = 36	6	6	0
Position = 37	9	3	1
Position = 38	11	7	1
Position = 39	6	3	0
Position = 40	5	0	0
Position = 41	8	4	0
Position = 42	14	12	4
Position = 43	10	6	0
Position = 44	8	6	0
Position = 45	14	7	1
Position = 46	9	3	2
Position = 47	3	2	0
Position = 48	16	7	1
Position = 49	6	4	0
Position = 50	9	2	0
Position = 51	15	8	1
Position = 52	6	3	1
Position = 53	4	1	0
Position = 54	8	3	0
Position = 55	11	5	1
Position = 56	12	6	0
Position = 57	12	4	2
Position = 58	11	4	2
Position = 59	8	3	2
Position = 60	10	9	0
Position = 61	13	8	1
Position = 62	14	8	2
Position = 63	13	8	4
Position = 64	11	6	0

Further analysis was applied to the results when more than k of the p-values were greater than k(.01)%. An upper 99% limit for the count when k = 10 is 17, and the corresponding values for k = 5 and k = 1 are: 10 and 3.

Counts falling above these values would be classified as significant.

The results of the frequency test on all bit positions in 64-bit subblocks show that bit positions 42 and 63 show some significance. This is not reflected in all three columns shown, and does not indicate a cause for concern.

The minimum p-value obtained over all 7,400 CRYPT-X tests was 0.0002.
The number of p-values below 0.1, 0.05 and 0.01 was 743, 375 and 74. These give proportions very close to their expected values of 0.1, 0.05 and 0.01 respectively.

Results of Statistical Randomness Tests on MUGI with Fixed Key & Variable IV

Length of output stream = 8,388,608 bits.
Length of output stream = 100,000 bits. (Complexity Tests)
Number of streams = 100
Sequence Complexity on 10^5 bits: Threshold value = 6,021 Mean value = 6,056

	Number of p-values below:		
Test	***.10***	***.05***	***.01***
Frequency	11	4	0
Binary Derivative (1)	8	4	1
Binary Derivative (2)	10	2	0
Change Point	22	7	2
Subblock b = 4	10	2	0
Subblock b = 8	15	8	2
Runs Distribution	6	2	0
Linear Complexity	2	2	1
LC profile - Jumps	16	10	3
LC Profile - Jump Size	19	10	2
Longest Run	Max = 30	Next = 28	
Sequence Complexity	Max = 6147	Min = 6056	

Further analysis was applied to the results when more than k of the p-values were greater than k(.01)%. An upper 99% limit for the count when k = 10 is calculated using:

$$100\left(0.1+2.326\sqrt{\frac{.1(1-.1)}{100}}\right) = 17$$

The corresponding values for k = 5 and k = 1 are: 10 and 3.
Counts falling above these values would be classified as significant.

The results for the Change Point test and the Linear Complexity Profile - Jump size are the only ones that show this significance.

Longest Run

For a random bit stream of length 2^{20}, the expected number of runs of length 30 ≈ $2^{20-30-1} = 2^{-11}$. This implies that it is highly unlikely that a run of length 30 will appear in a bit stream of this length.

The expected number of runs of length 22 ≈ $2^{20-22-1} = 2^{-3} \approx 0.125$, which supports that this is a more likely occurrence for the length of the longest run.

It would appear that the length of the longest run exceeds what would be expected in bit streams of this length. It should be noted that the length of the longest run may not exceed 30 for much longer streams. Hence we cannot conclude that this result shows any weakness in the MUGI algorithm.

Results of Frequency Test Applied to Bit Positions in 64-bit Subblocks
MUGI Keystreams with Fixed Key and Variable IV

Length of each MUGI keystream = 2^{20} bytes (8,388,608 bits).
Number of 64-bit subblocks = $2^{23}/2^{6} = 2^{17}$ bits (131,072 bits)
Number of keystreams for each test = 100

The table below gives the number of p-values falling below 0.1, 0.05 and 0.01 from the 100 MUGI keystreams tested. In a sample of 100 keystreams, the expected count for the number of p-values less than 0.1 is 10, for 0.05 is 5, and for 0.01 is 1.

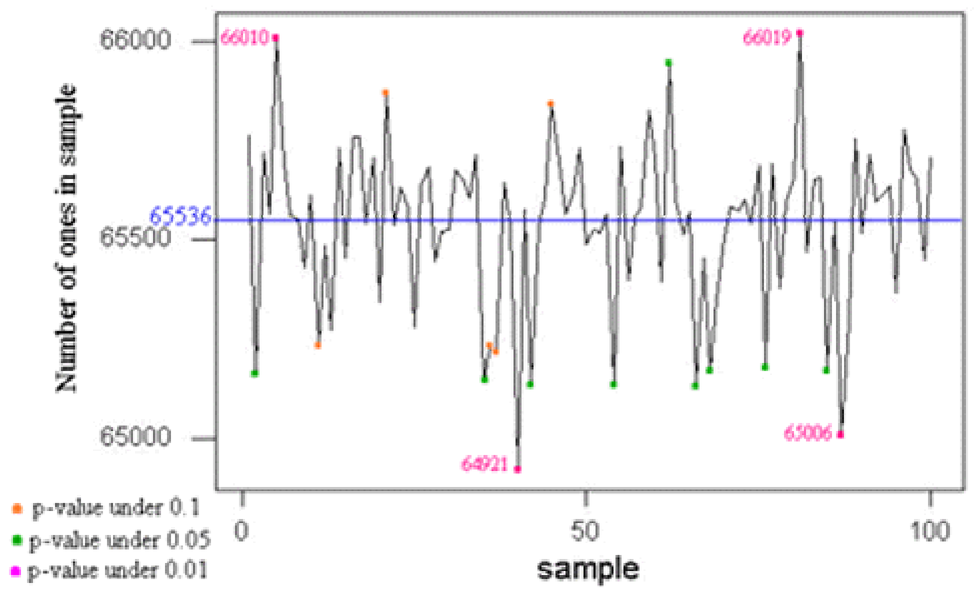
Bit Position in 64-bit Subblocks	***Number of p-values less than:***		
	0.1	***0.05***	***0.01***
Position = 1	12	4	0
Position = 2	12	4	1
Position = 3	3	1	0
Position = 4	12	7	2
Position = 5	13	5	1
Position = 6	10	6	1
Position = 7	9	6	1
Position = 8	4	2	0
Position = 9	12	4	0
Position = 10	9	3	0
Position = 11	12	9	0
Position = 12	7	3	0
Position = 13	9	3	0
Position = 14	18	12	4
Position = 15	15	8	4
Position = 16	9	4	0
Position = 17	11	7	2
Position = 18	11	4	1
Position = 19	10	4	2
Position = 20	9	5	2
Position = 21	14	8	1
Position = 22	5	2	1
Position = 23	4	1	0
Position = 24	6	1	0
Position = 25	13	9	3
Position = 26	9	6	0
Position = 27	10	4	2
Position = 28	7	5	2
Position = 29	10	8	2
Position = 30	11	5	3
Position = 31	8	5	2
Position = 32	10	5	1
Position = 33	13	7	0
Position = 34	8	3	1
Position = 35	8	1	0
Position = 36	6	3	1

Position = 37	11	5	1
Position = 38	12	5	1
Position = 39	11	5	1
Position = 40	11	2	0
Position = 41	13	10	3
Position = 42	7	4	1
Position = 43	10	7	4
Position = 44	14	11	2
Position = 45	13	4	0
Position = 46	8	4	1
Position = 47	5	3	0
Position = 48	7	6	0
Position = 49	12	5	0
Position = 50	11	5	2
Position = 51	9	5	2
Position = 52	8	3	0
Position = 53	8	6	0
Position = 54	7	3	1
Position = 55	13	3	1
Position = 56	11	3	0
Position = 57	11	6	1
Position = 58	10	4	1
Position = 59	13	6	0
Position = 60	12	10	2
Position = 61	9	3	0
Position = 62	12	7	2
Position = 63	13	4	3
Position = 64	9	3	1

Further analysis was applied to the results when more than k of the p-values were greater than k(.01)%. An upper 99% limit for the count when k = 10 is 17, and the corresponding values for k = 5 and k = 1 are: 10 and 3.

Counts falling above these values would be classified as significant.

The results of the frequency test on all bit positions in 64-bit subblocks show that bit positions 14, 15, 43 and 44 show some significance. This is only reflected in position 14 for all three columns shown. For further investigation of this result, a graph of the number of ones in the 100 MUGI keystream samples (with fixed key and variable IV) in position 14 of 64-bit subblocks is shown below.



The average number of ones in bit position 14 was 65,542. While this was higher than the expected value of 65,536 (half the number of bits) there were more of the 18 extreme values (those with p-values less than 0.1) falling below the mean. The number of ones have been inserted for the four samples with a p-value less than 0.01.

The minimum p-value obtained over all 7,400 CRYPT-X tests was 0.0001.
The number of p-values below 0.1, 0.05 and 0.01 was 758, 367 and 81. These give proportions very close to their expected values of 0.1, 0.05 and 0.01 respectively.

Appendix D. An Implementation of MUGI in ANSI C

This appendix contains code for the implementation of MUGI that is described in section 7.2.

D.1 The mugi.h header file

```
/**
 * @file mugi.h
 * Definitions for the MUGI PRNG
*/

#ifndef _MUGI_H_
#define _MUGI_H_

#ifdef __cplusplus
extern "C" {
#endif

#include "defs.h"

#ifdef _OPT_
extern word32 sbox[1024];
#else
extern word32 sbox[256];
#endif

/* Status codes */
#define MUGI_OK 0

/* Internals */
#define STATE_SIZE  3
#define BUFFER_SIZE 16

typedef struct
{
    int offset;
    word64 a[STATE_SIZE];
    word64 b[BUFFER_SIZE];

#ifndef _OPT_
    word64 C[3]; /* mugi constants */
#endif
} Mugi;

/**
 * Initialize the Mugi PRNG
 * @param  m    [In/Out]  mugi generator
 * @param  key  [In]      128 bit key
 * @param  IV   [In]      128 bit IV
 * @returns MUGI_OK on success
 */
int     Initialise(Mugi* m, const word128 key, const word128 IV);
/**
 * Extract a 64-bit word from the MUGI PRNG
 * @param  m [In/Out]  mugi generator
 * @returns pseudo-random word
 */
word64  Update(Mugi *m);

#ifdef __cplusplus
}
#endif
#endif
```

D.2 The mugi.c Source File

```
/**
 * @file mugi.c
 * Contains implementation of the MUGI PRNG
 */

#include "mugi.h"

#define ROL7(x)   (x << 7  | x >> 57)
#define ROL17(x)  (x << 17 | x >> 47)
#define ROL32(x)  (x << 32 | x >> 32)
#define ROL57(x)  (x << 57 | x >> 7)

#define XTIME(a) ((a & 0x80) ?  ((a << 1) ^ 0x11B) : (a << 1))

#define SBOX1(x, y) y ^= sbox[(x&0xFF)+768]; x>>=8;
#define SBOX2(x, y) y ^= sbox[(x&0xFF)+512]; x>>=8;
#define SBOX3(x, y) y ^= sbox[(x&0xFF)+256]; x>>=8;
#define SBOX4(x, y) y ^= sbox[(x&0xFF)]; x>>=8;

/* Pentium II: does not benefit from function->macro conversion */
word64 F(word64 a, word64 b)
{
    word64 x = a ^ b;
    word64 y = 0, y0, y1, z;

    SBOX1(x, y)
    SBOX2(x, y)
    SBOX3(x, y)
    SBOX4(x, y)

    y0 = y;
    y  = 0;

    SBOX1(x, y)
    SBOX2(x, y)
    SBOX3(x, y)
    SBOX4(x, y)

    y1 = y;
    z   = (y0 << 32 | y0) & 0xFFFF00000000FFFF;
    y1  = (y1 << 32 | y1) & 0x0000FFFFFFFF0000;
    z  |= y1;

    return z;
}
```

```c
#define _RHO(a, b4, b10, t0)  \
    t0   = a[2]; \
    a[2] = a[0] ^ F(a[1], ROL17(b10)) ^ 0x3C6EF372FE94F82B; \
    a[0] = a[1]; \
    a[1] = t0   ^ F(a[1], b4) ^ 0xBB67AE8584CAA73B;

#define OFFSET(base, offset)  ((base + offset) & 0xF)

#define _LAMBDA(b, a, offset) \
    offset--; \
    b[OFFSET(10, offset)] ^= ROL32(b[OFFSET(14, offset)]); \
    b[OFFSET(4, offset)] ^= b[OFFSET(8, offset)]; \
    b[OFFSET(0, offset)] ^= a;

/**
 * Initialize the Mugi PRNG
 * @param  m    [In/Out]  mugi generator
 * @param  key  [In]      128 bit key
 * @param  IV   [In]      128 bit IV
 * @returns MUGI_OK on success
 * @note not fully optimized
 */
int Initialise(Mugi *m, const word128 key, const word128 IV)
{
    word64 t0;
    int idx = 0;

    m->offset = 0;
    m->a[0] = key.h;
    m->a[1] = key.l;
    m->a[2] = ROL7(key.h) ^ ROL57(key.l) ^ 0x6A09E667F3BCC908;

    for (idx = 0; idx < 16; idx++) {
         _RHO(m->a, 0, 0, t0);
         m->b[OFFSET(15-idx, m->offset)] = m->a[0];
    }
    m->a[0] ^= IV.h;
    m->a[1] ^= IV.l;
    m->a[2] ^= ROL7(IV.h) ^ ROL57(IV.l) ^ 0x6A09E667F3BCC908;

    for (idx = 0; idx < 16; idx++) {
        _RHO(m->a, 0, 0, t0);
    }
    for (idx = 0; idx < 16; idx++) {
        Update(m);
    }
    return MUGI_OK;
}
```

```
/**
 * Extract a 64-bit word from the MUGI PRNG
 * @param  m [In/Out]  mugi generator
 * @returns pseudo-random word
 */
word64 Update(Mugi *m)
{
    /* proxying makes this much faster on the Intel PIII*/
    word64 b4  = m->b[OFFSET(4, m->offset)];
    word64 b10 = m->b[OFFSET(10, m->offset)];
    word64 res = m->a[2];
    word64 t0  = 0;

    _LAMBDA(m->b, m->a[0], m->offset);
    _RHO(m->a, b4, b10, t0);
    return res;
}
```

D.3 The sbox.c Source File

```
#include "defs.h"

word32 sbox[1024] =
{
    0xC66363A5, 0xF87C7C84, 0xEE777799, 0xF67B7B8D,
    0xFFF2F20D, 0xD66B6BBD, 0xDE6F6FB1, 0x91C5C554,
    0x60303050, 0x02010103, 0xCE6767A9, 0x562B2B7D,
    0xE7FEFE19, 0xB5D7D762, 0x4DABABE6, 0xEC76769A,
    0x8FCACA45, 0x1F82829D, 0x89C9C940, 0xFA7D7D87,
    0xEFFAFA15, 0xB25959EB, 0x8E4747C9, 0xFBF0F00B,
    0x41ADADEC, 0xB3D4D467, 0x5FA2A2FD, 0x45AFAFEA,
    0x239C9CBF, 0x53A4A4F7, 0xE4727296, 0x9BC0C05B,
    0x75B7B7C2, 0xE1FDFD1C, 0x3D9393AE, 0x4C26266A,
    0x6C36365A, 0x7E3F3F41, 0xF5F7F702, 0x83CCCC4F,
    0x6834345C, 0x51A5A5F4, 0xD1E5E534, 0xF9F1F108,
    0xE2717193, 0xABD8D873, 0x62313153, 0x2A15153F,
    0x0804040C, 0x95C7C752, 0x46232365, 0x9DC3C35E,
    0x30181828, 0x379696A1, 0x0A05050F, 0x2F9A9AB5,
    0x0E070709, 0x24121236, 0x1B80809B, 0xDFE2E23D,
    0xCDEBEB26, 0x4E272769, 0x7FB2B2CD, 0xEA75759F,
    0x1209091B, 0x1D83839E, 0x582C2C74, 0x341A1A2E,
    0x361B1B2D, 0xDC6E6EB2, 0xB45A5AEE, 0x5BA0A0FB,
    0xA45252F6, 0x763B3B4D, 0xB7D6D661, 0x7DB3B3CE,
    0x5229297B, 0xDDE3E33E, 0x5E2F2F71, 0x13848497,
    0xA65353F5, 0xB9D1D168, 0x00000000, 0xC1EDED2C,
    0x40202060, 0xE3FCFC1F, 0x79B1B1C8, 0xB65B5BED,
    0xD46A6ABE, 0x8DCBCB46, 0x67BEBED9, 0x7239394B,
    0x944A4ADE, 0x984C4CD4, 0xB05858E8, 0x85CFCF4A,
    0xBBD0D06B, 0xC5EFEF2A, 0x4FAAAAE5, 0xEDFBFB16,
    0x864343C5, 0x9A4D4DD7, 0x66333355, 0x11858594,
    0x8A4545CF, 0xE9F9F910, 0x04020206, 0xFE7F7F81,
    0xA05050F0, 0x783C3C44, 0x259F9FBA, 0x4BA8A8E3,
    0xA25151F3, 0x5DA3A3FE, 0x804040C0, 0x058F8F8A,
    0x3F9292AD, 0x219D9DBC, 0x70383848, 0xF1F5F504,
    0x63BCBCDF, 0x77B6B6C1, 0xAFDADA75, 0x42212163,
    0x20101030, 0xE5FFFF1A, 0xFDF3F30E, 0xBFD2D26D,
    0x81CDCD4C, 0x180C0C14, 0x26131335, 0xC3ECEC2F,
    0xBE5F5FE1, 0x359797A2, 0x884444CC, 0x2E171739,
    0x93C4C457, 0x55A7A7F2, 0xFC7E7E82, 0x7A3D3D47,
    0xC86464AC, 0xBA5D5DE7, 0x3219192B, 0xE6737395,
    0xC06060A0, 0x19818198, 0x9E4F4FD1, 0xA3DCDC7F,
    0x44222266, 0x542A2A7E, 0x3B9090AB, 0x0B888883,
    0x8C4646CA, 0xC7EEEE29, 0x6BB8B8D3, 0x2814143C,
    0xA7DEDE79, 0xBC5E5EE2, 0x160B0B1D, 0xADDBDB76,
    0xDBE0E03B, 0x64323256, 0x743A3A4E, 0x140A0A1E,
    0x924949DB, 0x0C06060A, 0x4824246C, 0xB85C5CE4,
    0x9FC2C25D, 0xBDD3D36E, 0x43ACACEF, 0xC46262A6,
    0x399191A8, 0x319595A4, 0xD3E4E437, 0xF279798B,
    0xD5E7E732, 0x8BC8C843, 0x6E373759, 0xDA6D6DB7,
    0x018D8D8C, 0xB1D5D564, 0x9C4E4ED2, 0x49A9A9E0,
    0xD86C6CB4, 0xAC5656FA, 0xF3F4F407, 0xCFEAEA25,
    0xCA6565AF, 0xF47A7A8E, 0x47AEAEE9, 0x10080818,
    0x6FBABAD5, 0xF0787888, 0x4A25256F, 0x5C2E2E72,
    0x381C1C24, 0x57A6A6F1, 0x73B4B4C7, 0x97C6C651,
    0xCBE8E823, 0xA1DDDD7C, 0xE874749C, 0x3E1F1F21,
    0x964B4BDD, 0x61BDBDDC, 0x0D8B8B86, 0x0F8A8A85,
    0xE0707090, 0x7C3E3E42, 0x71B5B5C4, 0xCC6666AA,
    0x904848D8, 0x06030305, 0xF7F6F601, 0x1C0E0E12,
    0xC26161A3, 0x6A35355F, 0xAE5757F9, 0x69B9B9D0,
    0x17868691, 0x99C1C158, 0x3A1D1D27, 0x279E9EB9,
    0xD9E1E138, 0xEBF8F813, 0x2B9898B3, 0x22111133,
    0xD26969BB, 0xA9D9D970, 0x078E8E89, 0x339494A7,
    0x2D9B9BB6, 0x3C1E1E22, 0x15878792, 0xC9E9E920,
    0x87CECE49, 0xAA5555FF, 0x50282878, 0xA5DFDF7A,
    0x038C8C8F, 0x59A1A1F8, 0x09898980, 0x1A0D0D17,
    0x65BFBFDA, 0xD7E6E631, 0x844242C6, 0xD06868B8,
    0x824141C3, 0x299999B0, 0x5A2D2D77, 0x1E0F0F11,
    0x7BB0B0CB, 0xA85454FC, 0x6DBBBBD6, 0x2C16163A,
    0xA5C66363, 0x84F87C7C, 0x99EE7777, 0x8DF67B7B,
    0x0DFFF2F2, 0xBDD66B6B, 0xB1DE6F6F, 0x5491C5C5,
    0x50603030, 0x03020101, 0xA9CE6767, 0x7D562B2B,
    0x19E7FEFE, 0x62B5D7D7, 0xE64DABAB, 0x9AEC7676,
    0x458FCACA, 0x9D1F8282, 0x4089C9C9, 0x87FA7D7D,
    0x15EFFAFA, 0xEBB25959, 0xC98E4747, 0x0BFBF0F0,
    0xEC41ADAD, 0x67B3D4D4, 0xFD5FA2A2, 0xEA45AFAF,
    0xBF239C9C, 0xF753A4A4, 0x96E47272, 0x5B9BC0C0,
    0xC275B7B7, 0x1CE1FDFD, 0xAE3D9393, 0x6A4C2626,
    0x5A6C3636, 0x417E3F3F, 0x02F5F7F7, 0x4F83CCCC,
    0x5C683434, 0xF451A5A5, 0x34D1E5E5, 0x08F9F1F1,
    0x93E27171, 0x73ABD8D8, 0x53623131, 0x3F2A1515,
    0x0C080404, 0x5295C7C7, 0x65462323, 0x5E9DC3C3,
    0x28301818, 0xA1379696, 0x0F0A0505, 0xB52F9A9A,
    0x090E0707, 0x36241212, 0x9B1B8080, 0x3DDFE2E2,
    0x26CDEBEB, 0x694E2727, 0xCD7FB2B2, 0x9FEA7575,
    0x1B120909, 0x9E1D8383, 0x74582C2C, 0x2E341A1A,
    0x2D361B1B, 0xB2DC6E6E, 0xEEB45A5A, 0xFB5BA0A0,
    0xF6A45252, 0x4D763B3B, 0x61B7D6D6, 0xCE7DB3B3,
    0x7B522929, 0x3EDDE3E3, 0x715E2F2F, 0x97138484,
    0xF5A65353, 0x68B9D1D1, 0x00000000, 0x2CC1EDED,
    0x60402020, 0x1FE3FCFC, 0xC879B1B1, 0xEDB65B5B,
    0xBED46A6A, 0x468DCBCB, 0xD967BEBE, 0x4B723939,
    0xDE944A4A, 0xD4984C4C, 0xE8B05858, 0x4A85CFCF,
    0x6BBBD0D0, 0x2AC5EFEF, 0xE54FAAAA, 0x16EDFBFB,
    0xC5864343, 0xD79A4D4D, 0x55663333, 0x94118585,
    0xCF8A4545, 0x10E9F9F9, 0x06040202, 0x81FE7F7F,
    0xF0A05050, 0x44783C3C, 0xBA259F9F, 0xE34BA8A8,
    0xF3A25151, 0xFE5DA3A3, 0xC0804040, 0x8A058F8F,
    0xAD3F9292, 0xBC219D9D, 0x48703838, 0x04F1F5F5,
    0xDF63BCBC, 0xC177B6B6, 0x75AFDADA, 0x63422121,
    0x30201010, 0x1AE5FFFF, 0x0EFDF3F3, 0x6DBFD2D2,
    0x4C81CDCD, 0x14180C0C, 0x35261313, 0x2FC3ECEC,
    0xE1BE5F5F, 0xA2359797, 0xCC884444, 0x392E1717,
    0x5793C4C4, 0xF255A7A7, 0x82FC7E7E, 0x477A3D3D,
    0xACC86464, 0xE7BA5D5D, 0x2B321919, 0x95E67373,
    0xA0C06060, 0x98198181, 0xD19E4F4F, 0x7FA3DCDC,
    0x66442222, 0x7E542A2A, 0xAB3B9090, 0x830B8888,
    0xCA8C4646, 0x29C7EEEE, 0xD36BB8B8, 0x3C281414,
    0x79A7DEDE, 0xE2BC5E5E, 0x1D160B0B, 0x76ADDBDB,
    0x3BDBE0E0, 0x56643232, 0x4E743A3A, 0x1E140A0A,
    0xDB924949, 0x0A0C0606, 0x6C482424, 0xE4B85C5C,
    0x5D9FC2C2, 0x6EBDD3D3, 0xEF43ACAC, 0xA6C46262,
    0xA8399191, 0xA4319595, 0x37D3E4E4, 0x8BF27979,
    0x32D5E7E7, 0x438BC8C8, 0x596E3737, 0xB7DA6D6D,
    0x8C018D8D, 0x64B1D5D5, 0xD29C4E4E, 0xE049A9A9,
    0xB4D86C6C, 0xFAAC5656, 0x07F3F4F4, 0x25CFEAEA,
    0xAFCA6565, 0x8EF47A7A, 0xE947AEAE, 0x18100808,
    0xD56FBABA, 0x88F07878, 0x6F4A2525, 0x725C2E2E,
    0x24381C1C, 0xF157A6A6, 0xC773B4B4, 0x5197C6C6,
    0x23CBE8E8, 0x7CA1DDDD, 0x9CE87474, 0x213E1F1F,
    0xDD964B4B, 0xDC61BDBD, 0x860D8B8B, 0x850F8A8A,
    0x90E07070, 0x427C3E3E, 0xC471B5B5, 0xAACC6666,
    0xD8904848, 0x05060303, 0x01F7F6F6, 0x121C0E0E,
    0xA3C26161, 0x5F6A3535, 0xF9AE5757, 0xD069B9B9,
    0x91178686, 0x5899C1C1, 0x273A1D1D, 0xB9279E9E,
    0x38D9E1E1, 0x13EBF8F8, 0xB32B9898, 0x33221111,
    0xBBD26969, 0x70A9D9D9, 0x89078E8E, 0xA7339494,
    0xB62D9B9B, 0x223C1E1E, 0x92158787, 0x20C9E9E9,
    0x4987CECE, 0xFFAA5555, 0x78502828, 0x7AA5DFDF,
    0x8F038C8C, 0xF859A1A1, 0x80098989, 0x171A0D0D,
    0xDA65BFBF, 0x31D7E6E6, 0xC6844242, 0xB8D06868,
    0xC3824141, 0xB0299999, 0x775A2D2D, 0x111E0F0F,
    0xCB7BB0B0, 0xFCA85454, 0xD66DBBBB, 0x3A2C1616,
    0x63A5C663, 0x7C84F87C, 0x7799EE77, 0x7B8DF67B,
    0xF20DFFF2, 0x6BBDD66B, 0x6FB1DE6F, 0xC55491C5,
    0x30506030, 0x01030201, 0x67A9CE67, 0x2B7D562B,
    0xFE19E7FE, 0xD762B5D7, 0xABE64DAB, 0x769AEC76,
    0xCA458FCA, 0x829D1F82, 0xC94089C9, 0x7D87FA7D,
    0xFA15EFFA, 0x59EBB259, 0x47C98E47, 0xF00BFBF0,
    0xADEC41AD, 0xD467B3D4, 0xA2FD5FA2, 0xAFEA45AF,
    0x9CBF239C, 0xA4F753A4, 0x7296E472, 0xC05B9BC0,
    0xB7C275B7, 0xFD1CE1FD, 0x93AE3D93, 0x266A4C26,
    0x365A6C36, 0x3F417E3F, 0xF702F5F7, 0xCC4F83CC,
    0x345C6834, 0xA5F451A5, 0xE534D1E5, 0xF108F9F1,
    0x7193E271, 0xD873ABD8, 0x31536231, 0x153F2A15,
    0x040C0804, 0xC75295C7, 0x23654623, 0xC35E9DC3,
    0x18283018, 0x96A13796, 0x050F0A05, 0x9AB52F9A,
    0x07090E07, 0x12362412, 0x809B1B80, 0xE23DDFE2,
    0xEB26CDEB, 0x27694E27, 0xB2CD7FB2, 0x759FEA75,
    0x091B1209, 0x839E1D83, 0x2C74582C, 0x1A2E341A,
    0x1B2D361B, 0x6EB2DC6E, 0x5AEEB45A, 0xA0FB5BA0,
    0x52F6A452, 0x3B4D763B, 0xD661B7D6, 0xB3CE7DB3,
    0x297B5229, 0xE33EDDE3, 0x2F715E2F, 0x84971384,
    0x53F5A653, 0xD168B9D1, 0x00000000, 0xED2CC1ED,
    0x20604020, 0xFC1FE3FC, 0xB1C879B1, 0x5BEDB65B,
    0x6ABED46A, 0xCB468DCB, 0xBED967BE, 0x394B7239,
    0x4ADE944A, 0x4CD4984C, 0x58E8B058, 0xCF4A85CF,
    0xD06BBBD0, 0xEF2AC5EF, 0xAAE54FAA, 0xFB16EDFB,
    0x43C58643, 0x4DD79A4D, 0x33556633, 0x85941185,
    0x45CF8A45, 0xF910E9F9, 0x02060402, 0x7F81FE7F,
    0x50F0A050, 0x3C44783C, 0x9FBA259F, 0xA8E34BA8,
    0x51F3A251, 0xA3FE5DA3, 0x40C08040, 0x8F8A058F,
    0x92AD3F92, 0x9DBC219D, 0x38487038, 0xF504F1F5,
    0xBCDF63BC, 0xB6C177B6, 0xDA75AFDA, 0x21634221,
    0x10302010, 0xFF1AE5FF, 0xF30EFDF3, 0xD26DBFD2,
    0xCD4C81CD, 0x0C14180C, 0x13352613, 0xEC2FC3EC,
    0x5FE1BE5F, 0x97A23597, 0x44CC8844, 0x17392E17,
    0xC45793C4, 0xA7F255A7, 0x7E82FC7E, 0x3D477A3D,
    0x64ACC864, 0x5DE7BA5D, 0x192B3219, 0x7395E673,
    0x60A0C060, 0x81981981, 0x4FD19E4F, 0xDC7FA3DC,
    0x22664422, 0x2A7E542A, 0x90AB3B90, 0x88830B88,
    0x46CA8C46, 0xEE29C7EE, 0xB8D36BB8, 0x143C2814,
    0xDE79A7DE, 0x5EE2BC5E, 0x0B1D160B, 0xDB76ADDB,
    0xE03BDBE0, 0x32566432, 0x3A4E743A, 0x0A1E140A,
    0x49DB9249, 0x060A0C06, 0x246C4824, 0x5CE4B85C,
    0xC25D9FC2, 0xD36EBDD3, 0xACEF43AC, 0x62A6C462,
    0x91A83991, 0x95A43195, 0xE437D3E4, 0x798BF279,
    0xE732D5E7, 0xC8438BC8, 0x37596E37, 0x6DB7DA6D,
    0x8D8C018D, 0xD564B1D5, 0x4ED29C4E, 0xA9E049A9,
    0x6CB4D86C, 0x56FAAC56, 0xF407F3F4, 0xEA25CFEA,
    0x65AFCA65, 0x7A8EF47A, 0xAEE947AE, 0x08181008,
    0xBAD56FBA, 0x7888F078, 0x256F4A25, 0x2E725C2E,
    0x1C24381C, 0xA6F157A6, 0xB4C773B4, 0xC65197C6,
    0xE823CBE8, 0xDD7CA1DD, 0x749CE874, 0x1F213E1F,
    0x4BDD964B, 0xBDDC61BD, 0x8B860D8B, 0x8A850F8A,
    0x7090E070, 0x3E427C3E, 0xB5C471B5, 0x66AACC66,
    0x48D89048, 0x03050603, 0xF601F7F6, 0x0E121C0E,
    0x61A3C261, 0x355F6A35, 0x57F9AE57, 0xB9D069B9,
    0x86911786, 0xC15899C1, 0x1D273A1D, 0x9EB9279E,
    0xE138D9E1, 0xF813EBF8, 0x98B32B98, 0x11332211,
    0x69BBD269, 0xD970A9D9, 0x8E89078E, 0x94A73394,
    0x9BB62D9B, 0x1E223C1E, 0x87921587, 0xE920C9E9,
    0xCE4987CE, 0x55FFAA55, 0x28785028, 0xDF7AA5DF,
    0x8C8F038C, 0xA1F859A1, 0x89800989, 0x0D171A0D,
    0xBFDA65BF, 0xE631D7E6, 0x42C68442, 0x68B8D068,
    0x41C38241, 0x99B02999, 0x2D775A2D, 0x0F111E0F,
    0xB0CB7BB0, 0x54FCA854, 0xBBD66DBB, 0x163A2C16,
    0x6363A5C6, 0x7C7C84F8, 0x777799EE, 0x7B7B8DF6,
    0xF2F20DFF, 0x6B6BBDD6, 0x6F6FB1DE, 0xC5C55491,
    0x30305060, 0x01010302, 0x6767A9CE, 0x2B2B7D56,
    0xFEFE19E7, 0xD7D762B5, 0xABABE64D, 0x76769AEC,
    0xCACA458F, 0x82829D1F, 0xC9C94089, 0x7D7D87FA,
    0xFAFA15EF, 0x5959EBB2, 0x4747C98E, 0xF0F00BFB,
    0xADADEC41, 0xD4D467B3, 0xA2A2FD5F, 0xAFAFEA45,
    0x9C9CBF23, 0xA4A4F753, 0x727296E4, 0xC0C05B9B,
    0xB7B7C275, 0xFDFD1CE1, 0x9393AE3D, 0x26266A4C,
    0x36365A6C, 0x3F3F417E, 0xF7F702F5, 0xCCCC4F83,
    0x34345C68, 0xA5A5F451, 0xE5E534D1, 0xF1F108F9,
    0x717193E2, 0xD8D873AB, 0x31315362, 0x15153F2A,
    0x04040C08, 0xC7C75295, 0x23236546, 0xC3C35E9D,
    0x18182830, 0x9696A137, 0x05050F0A, 0x9A9AB52F,
    0x0707090E, 0x12123624, 0x80809B1B, 0xE2E23DDF,
    0xEBEB26CD, 0x2727694E, 0xB2B2CD7F, 0x75759FEA,
    0x09091B12, 0x83839E1D, 0x2C2C7458, 0x1A1A2E34,
    0x1B1B2D36, 0x6E6EB2DC, 0x5A5AEEB4, 0xA0A0FB5B,
    0x5252F6A4, 0x3B3B4D76, 0xD6D661B7, 0xB3B3CE7D,
    0x29297B52, 0xE3E33EDD, 0x2F2F715E, 0x84849713,
    0x5353F5A6, 0xD1D168B9, 0x00000000, 0xEDED2CC1,
    0x20206040, 0xFCFC1FE3, 0xB1B1C879, 0x5B5BEDB6,
    0x6A6ABED4, 0xCBCB468D, 0xBEBED967, 0x39394B72,
    0x4A4ADE94, 0x4C4CD498, 0x5858E8B0, 0xCFCF4A85,
    0xD0D06BBB, 0xEFEF2AC5, 0xAAAAE54F, 0xFBFB16ED,
    0x4343C586, 0x4D4DD79A, 0x33335566, 0x85859411,
    0x4545CF8A, 0xF9F910E9, 0x02020604, 0x7F7F81FE,
    0x5050F0A0, 0x3C3C4478, 0x9F9FBA25, 0xA8A8E34B,
    0x5151F3A2, 0xA3A3FE5D, 0x4040C080, 0x8F8F8A05,
    0x9292AD3F, 0x9D9DBC21, 0x38384870, 0xF5F504F1,
    0xBCBCDF63, 0xB6B6C177, 0xDADA75AF, 0x21216342,
    0x10103020, 0xFFFF1AE5, 0xF3F30EFD, 0xD2D26DBF,
    0xCDCD4C81, 0x0C0C1418, 0x13133526, 0xECEC2FC3,
    0x5F5FE1BE, 0x9797A235, 0x4444CC88, 0x1717392E,
    0xC4C45793, 0xA7A7F255, 0x7E7E82FC, 0x3D3D477A,
    0x6464ACC8, 0x5D5DE7BA, 0x19192B32, 0x737395E6,
    0x6060A0C0, 0x81819819, 0x4F4FD19E, 0xDCDC7FA3,
    0x22226644, 0x2A2A7E54, 0x9090AB3B, 0x8888830B,
    0x4646CA8C, 0xEEEE29C7, 0xB8B8D36B, 0x14143C28,
    0xDEDE79A7, 0x5E5EE2BC, 0x0B0B1D16, 0xDBDB76AD,
    0xE0E03BDB, 0x32325664, 0x3A3A4E74, 0x0A0A1E14,
    0x4949DB92, 0x06060A0C, 0x24246C48, 0x5C5CE4B8,
    0xC2C25D9F, 0xD3D36EBD, 0xACACEF43, 0x6262A6C4,
    0x9191A839, 0x9595A431, 0xE4E437D3, 0x79798BF2,
    0xE7E732D5, 0xC8C8438B, 0x3737596E, 0x6D6DB7DA,
    0x8D8D8C01, 0xD5D564B1, 0x4E4ED29C, 0xA9A9E049,
    0x6C6CB4D8, 0x5656FAAC, 0xF4F407F3, 0xEAEA25CF,
    0x6565AFCA, 0x7A7A8EF4, 0xAEAEE947, 0x08081810,
    0xBABAD56F, 0x787888F0, 0x25256F4A, 0x2E2E725C,
    0x1C1C2438, 0xA6A6F157, 0xB4B4C773, 0xC6C65197,
    0xE8E823CB, 0xDDDD7CA1, 0x74749CE8, 0x1F1F213E,
    0x4B4BDD96, 0xBDBDDC61, 0x8B8B860D, 0x8A8A850F,
    0x707090E0, 0x3E3E427C, 0xB5B5C471, 0x6666AACC,
    0x4848D890, 0x03030506, 0xF6F601F7, 0x0E0E121C,
    0x6161A3C2, 0x35355F6A, 0x5757F9AE, 0xB9B9D069,
    0x86869117, 0xC1C15899, 0x1D1D273A, 0x9E9EB927,
    0xE1E138D9, 0xF8F813EB, 0x9898B32B, 0x11113322,
    0x6969BBD2, 0xD9D970A9, 0x8E8E8907, 0x9494A733,
    0x9B9BB62D, 0x1E1E223C, 0x87879215, 0xE9E920C9,
    0xCECE4987, 0x5555FFAA, 0x28287850, 0xDFDF7AA5,
    0x8C8C8F03, 0xA1A1F859, 0x89898009, 0x0D0D171A,
    0xBFBFDA65, 0xE6E631D7, 0x4242C684, 0x6868B8D0,
    0x4141C382, 0x9999B029, 0x2D2D775A, 0x0F0F111E,
    0xB0B0CB7B, 0x5454FCA8, 0xBBBBD66D, 0x16163A2C
};
```