Evaluation of TOYOCRYPT-HR1

January 17, 2001

Information Security Research Centre
Queensland University of Technology
Ed Dawson, Andrew Clark, Helen Gustafson,
Bill Millan, Leonie Simpson

TABLE OF CONTENTS

Evaluation of TOYOCRYPT-HR1 Algorithm

1 Executive Summary

This is a report on the analysis of the Pseudorandom Number Generator (PRNG), TOYOCRYPT-HR1. For this algorithm the evaluators have

- (i) analysed structural aspects;
- (ii) evaluated basic cryptographic properties;
- (iii) measured the entropy;
- (iv) evaluated security from attack;
- (v) evaluated statistical properties;
- (vi) surveyed the speed.

The TOYOCRYPT-HR1 algorithm is a standard design for a PRNG using a linear feedback shift register (LFSR) together with a nonlinear Boolean function. This design produces a sequence with provable security properties in relation to period and high linear complexity. The sequences analysed displayed white-noise statistics and high entropy. The algorithm is secure from standard cryptographic attacks including linear cryptanalysis and correlation attacks. However, the evaluators have identified three different flaws with the algorithm.

Flaw 1 Divide and Conquer Attack

There is a possible divide and conquer attack which is significantly faster than exhaustive key search. We have described a simple method to change the algorithm to correct this flaw.

Flaw 2 Operational Speed in Software

The algorithm is very slow in software for many applications. The designers of the algorithm have recognised this problem and have included a parallel mode of operation to increase speed. However, it should be noted that this will also increase the key length and further this adaption can be applied to most PRNG's.

Flaw 3 Key Generation

The process of using the key to generate a new primitive polynomial is very slow for many applications. The evaluators are of the opinion that due to this problem a fixed LFSR will need to be used in many applications.

2 Description of TOYOCRYPT-HR1

The TOYOCRYPT-HR1 pseudorandom sequence generator consists of a single regularly clocked 128-bit linear feedback shift register (LFSR) and a nonlinear Boolean function, f, of 127 input variables. The function f is used as a nonlinear filter function, and the generator is known as a nonlinear filter generator. The algebraic normal form of function f is given in Appendix B. The keystream is generated by applying the nonlinear function f to the contents of 127 of the 128 stages of the LFSR. Each time the LFSR is clocked, an output bit is produced.

The TOYOCRYPT-HR1 keystream generator has two 128-bit keys: a fixed key and a stream key. The fixed key consists of the coefficients of the LFSR characteristic polynomial, and is selected such that the characteristic polynomial is primitive and irreducible on F_2. There are approximately 2^{120} such polynomials. The stream key is used to form the LFSR initial state, and is an arbitrary 128 bits, except for the all zero initial state. Thus there are $2^{128} - 1$ possible stream keys. Hence, the effective key size is 248.

2.1 Notation

Let the LFSR output sequence be denoted $d = \{d(t)\}_{t=0}^{\infty}$. Let the contents of the 128 stages of the LFSR at time t be denoted $x_0(t)$, $x_1(t)$, … , $x_{127}(t)$. Then $d(t) = x_0(t-1)$, for $t \geq 1$. Denote the sequence of inputs to the filter function f as $X = \{X(t)\}_{t=0}^{\infty}$ where $X(t) = \{x_0(t), x_1(t), \ldots, x_{125}(t), x_{127}(t)\}$. Note that $x_{126}(t)$ is not used as input to f. Then the output sequence of the nonlinear filter generator is the keystream $z = \{z(t)\}_{t=0}^{\infty}$, where $z(t) = f(X(t))$, for $t \geq 0$.

3 Structural Aspects

3.1 LFSR Operation

The LFSR uses a Galois configuration rather than the more standard Fibonacci configuration. The Galois configuration offers a significant increase in speed over the Fibonacci configuration in both software and hardware. In software we would estimate on the average about a four to five times improvement in speed. The speed of an LFSR implemented using the Galois configuration is independent of the number of taps in the feedback polynomial since it does not require individual bits to be extracted for use in calculating the next output bit. The Galois configuration also has the effect that LFSR state bits change in time (due to XOR with the feedback bit).

3.2 Analysis of the nonlinear function

Linear feedback shift registers are commonly used in pseudorandom number generators, as the properties of sequences produced by LFSRs are well known, their implementation in hardware or software is easy, and they allow for fast encryption and decryption rates. However, LFSR sequences are easily predicted from a short segment. For an LFSR of length 128, the entire period of the LFSR output sequence can be determined from only 128 successive terms in the sequence if the feedback polynomial is known, and from only 256 successive bits if the feedback polynomial is unknown [MASS 69]. This vulnerability to known plaintext attacks makes LFSRs unsuitable for use as pseudorandom sequence generators on their own. One way to make use of the good properties of LFSR sequences while avoiding the weakness due to their linearity is to introduce nonlinearity by using a nonlinear Boolean function operating on the contents of several stages of a regularly clocked LFSR. The choice of Boolean function used contributes significantly to the cryptographic security of the pseudorandom sequence generator. In this section, the Boolean function *f*, used as nonlinear combining function in the TOYOCRYPT-HR1 pseudorandom sequence generator, is examined.

The nonlinear feedforward function *f* is a Boolean function that takes as input the contents of 127 of the 128 stages of the LFSR and produces a single output bit. In [SPEC HR1] it is described as $f : F_2^{128} \to F_2$, although, as $x_{126}(t)$ is not used in *f*, it is more properly described as $f : F_2^{127} \to F_2$. For simplicity, in this section of the report, denote the contents of the stages of the LFSR as $x_0, x_1, x_2, \ldots x_{127}$. (that is, drop the (t)). As in [SPEC HR1], consider the following division of the LFSR: $X = \{x_{127}, x_{126}, X_L, X_R,\}$ where $X_L = \{x_{125}, x_{124}, \ldots, x_{63}\}$ and $X_R = \{x_{62}, x_{61}, \ldots, x_0\}$.

The function *f* consists of a single linear term, sixty-three quadratic terms, and three other terms of orders four, seventeen and sixty-three, respectively. The linear term is x_{127}. The quadratic terms are products, each with one factor from X_L and the other from X_R. Note that all terms of order greater than two are products for which all factors are stages within X_R. This is a potential weakness, which may be exploited in a divide and conquer style attack (see Section 5.2).

A basic condition for nonlinear filter generators for cryptographic applications is that the output sequence is balanced. This is true only if the combining function *f* is balanced. As the function *f* is linear in the variable x_{127}, then it is balanced. Thus the symbol frequency of the output sequence, *z*, is the same as for the underlying LFSR sequence, *d*.

The nonlinearity of a Boolean function is defined to be the minimum Hamming distance between the function and any affine function. If the nonlinearity is low, then the function is closely approximated by an affine function, a weakness which is exploited by fast correlation attacks. For Boolean functions with an even number of inputs, *n*, the maximum nonlinearity is known to be $2^{n-1} - 2^{\frac{n}{2}-1}$, and is obtained by a class of Boolean function known as bent functions. The subfunction *h* is a bent function (note that $f = h + x_{127}$) of 126 variables, with nonlinearity $N_h = 2^{125} - 2^{62}$ (see [ROTH 76]). Bent functions are not balanced, so are unsuitable for use as combining functions. The function *f* has a bent subfunction, to provide high nonlinearity, but has an additional linear term to provide balance. For the function *f*, with $n = 127$, the nonlinearity is $2^{126} - 2^{63}$. This implies that the bias available to any linear correlation is upper bounded by 2^{-64}. This value is sufficiently low to provide security (see Section 5).

Further examination of the quadratic terms of the function *f* was performed. For each quadratic term $x_i x_j$, $i \in X_R$ and $j \in X_L$, the difference $j-i$ was calculated. The differences range in value from 17 (for $x_{62}x_{79}$) to 102 (for $x_{17}x_{119}$ and $x_{19}x_{121}$). Interestingly, as illustrated by the last example, the differences are not necessarily unique. Differences of 34, 48, 53, 56,66,69,70,89,91,96 and 102 each occur for two terms, the difference 50 occurs for three terms, and the difference 72 occurs for 4 terms (see Appendix C). The existence of multiple quadratic terms with common differences is a potential weakness. As an example to illustrate the way this can be exploited, consider the two quadratic terms x_3x_{75} and x_6x_{78}. Whatever the value of the term x_3x_{75}, after clocking the LFSR three times, this value is now held by x_6x_{78} (with due allowance for the Galois configuration of the LFSR). There are sixty-three quadratic terms, so the proportion of quadratic terms with common differences is quite small, and perhaps cannot be fully exploited in an attack. However, changing the arrangement of the quadratic terms in the filter function could eliminate this potential weakness entirely. A simple way to do this is to select a different 63 bit permutation.

3.3 Key Management

There is a 256 bit key which has two components. One component is 128 bits, called the stream key, which is used as initial state vector for LFSR. A second component is 128 bits, called the fixed key, which is used to generate primitive polynomials to define tap settings for LFSR using Algorithm 4.6 from [SPEC HS1]. This process is very slow (see Section 6.3) since many different fixed keys would need to be tested before a primitive polynomial is found. The time taken for this process may cause problems in certain applications as explained further below.

In certain applications it is possible to overcome this problem with having to check multiple fixed keys by having the primitive polynomial generated offline. This process works well in key management processes called key transport where one party in the communication process actually creates keys. However there are many key management protocols where this is not possible. For example, in the key agreement process which uses the Diffie-Hellman Key Agreement algorithm each party in a key generation process inputs a random number. These are combined together to form the actual key using exponentiation process. This process in most applications needs to be done online. This is very costly in terms of time and communication bandwidth if this process is required to be repeated multiple times.

It is the opinion of the evaluators, due to the above problem with the generation of primitive polynomials, that in many applications a fixed LFSR will be used. This reduces the key to the 128 bit initial state vector of LFSR.

3.4 Modes of Operation

The TOYOCRYPT-HR1 algorithm is especially slow in software. In order to overcome this problem the designers have included a parallel mode of operation. This involves the generation of n independent sequences in parallel (see [SPEC HR1]) where n is less than or equal to the word size of the processor. This leads to higher speed for keystream generation (see Section 6.1). There is nothing unique about this process. This method can be applied to almost any keystream generator. However, it should be noted that a much larger key size is required for this parallel mode.

4 Keystream Properties

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

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

4.1 Period

The keystream properties cited in [EVAL HR1] relating to period were verified. That is, for the TOYOCRYPT-HR1 keystream generator, the period of the sequence produced by the keystream generator is $P_z = 2^{128} - 1$.

4.2 Linear Complexity

The linear complexity test checks for the minimum amount of knowledge required to reconstruct the whole stream using a linear feedback shift register. It is difficult to determine exactly the linear complexity of a sequence from TOYOCRYPT-HR1 equal in length to the complete period. However, bounds for this linear complexity can be found. The evaluators are in agreement with the results in [EVAL HR1] that the linear complexity of a keystream sequence of length P_z has a lower and upper bound of 2^{124} and 2^{127}. The selection of higher products of degree 2, 4, 17 and 63 means that the linear complexity profile of the keystream should approximately follow the n/2 line for strings of length n.

In order to obtain empirical evidence for the linear complexity and linear complexity profile we applied the tests from the CRYPT-X package. These linear complexity tests were applied to ten TOYOCRYPT-HR1 output bit-streams each of length 10^6 bits. The results gave linear complexity values extremely close to that expected for random data (i.e. half the bit-stream length). For more detailed results see Appendix D. 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 demonstrate 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 Statistical Analysis

A preliminary statistical analysis was carried out on TOYOCRYPT-HR1, in order to confirm the results in [EVAL HR1]. This analysis included three tests, namely a bit-frequency test, a subblock test on 4-bit subblocks, and a runs distribution (including the longest run). The tests were applied to 100 streams of length 20,000 bits each. The test results show that 9 of the 300 tests (or 0.03) gave p-values below 0.05. These results support the randomness of the TOYOCRYPT-HR1 output. The evaluators confirmed these results using the CRYPT-X package.

Further statistical analysis was carried out by the evaluators. The above tests plus additional statistical randomness tests from CRYPT-X were applied to much longer output streams from TOYOCRYPT-HR1. A further measure of sequence complexity was added to check the period of the output stream. The tests are based on the hypothesis that the measure obtained from the output stream supports randomness. The p-values obtained for the tests represent the probability that such a sample result would be obtained if the algorithm produces a random stream. Very small p-values would support non-randomness.

The sequence complexity test was applied to binary output streams of 10^6 bits (due to the amount of time required for the test), using ten different keys. The remaining tests were applied to ten binary output streams of 10^7 bits.

The subblock tests were applied to the output stream by dividing the bit-stream into non-overlapping subblocks of lengths ranging from 2 to 30 bits. The maximum subblock length of 30 was determined from the length of the file and the limitations of the test applied.

Statistical Analysis Results
The results of the CRYPT-X tests are summarised using p-values in Appendix E. The lowest p-value for any one test is 0.0007, with 24 of the 340 p-values obtained falling below 0.05. This represents a proportion of 0.07 of the tests applied, which is above the 0.05 level, yet not significant. Hence these results supports the randomness of the output from TOYOCRYPT-HR1.

4.4 Entropy

We can measure the entropy of TOYOCRYPT-HR1 in relation to the entropy of the input sequence and the output sequence.

4.4.1 Entropy of Input Sequence

The entropy of input sequence relates to the uncertainty of finding the state vector of the LFSR. For discussions on how difficult this is, we refer to Section 5 of this report on possible attacks, since the aim of most of these attacks is to find this state vector in less than exhaustive search.

4.4.2 Entropy of Output Sequence

In order to measure the entropy of the output sequence of TOYOCRYPT-HR1 we divided the output of sequence into subblocks. The analysis of subblock patterns is an important measure of the entropy of sequences produced by PRNG’s. The test applied gives the entropy of a b-bit subblock as the number of independent bits per subblock generated. Additional tests include the distribution of ones in these b-bit subblocks and the distribution of ones in the binary derivative of these blocks. For further explanation of these tests see the CRYPT-X manual (Appendix A).

These tests were applied to one large sample of 1 Gigabyte of output from TOYOCRYPT-HR1. The entropy test was applied to subblocks of length 48. No larger subblock lengths could be taken, as a much larger file would be required, that exceeded the storage limitation of the computer.

The CRYPT-X package requires a considerably longer output stream than 1 Gigabyte to investigate the entropy of subblocks larger than 48 bits, as it will only break larger subblocks into 48-bit blocks and give no further information than what has been obtained on the 48-bit subblock entropy value. To investigate larger subblocks alternative tests were applied with the subblock length ranging from 56 to 1024. These tests were based on the hypothesis that the ones are distributed independently and symmetrically, and the hypothesis that the runs are distributed independently and symmetrically, in the subblock.

Entropy Tests Results
For a bit-stream of 1 Gigabyte the longest subblock that could be tested using the 'repetition test' applied was 48 bits. Results give a strong emphasis that full entropy exists. This means that there are 48 independent bits per 48-bit block, and so any set of 48-bit subblocks obtained from a TOYOCRYPT-HR1 output stream will appear to be randomly generated.

For subblocks lengths from 56 to 1024 the distribution of ones and the distribution of runs (binary derivative) in the b-bit subblock are distributed independently and symmetrically (see Appendix F). This gives support to maximum entropy of the output from TOYOCRYPT-HR1 for subblocks of length from 56 to 1024.

5 Possible Attacks

In this section of the report, possible known plaintext attacks on the nonlinear filter generator are investigated. It is assumed in the discussion that the structure of the generator is known to an attacker, and that only the keys are unknown.

For most of these attacks, the discussion assumes that the fixed key (the LFSR feedback function) is known, and only the stream key (that is, the initial state of the LFSR) is unknown. Such a situation is possible, as it is likely that the fixed key can be obtained through reverse engineering (if in hardware), or by examining source code (if in software). Note that if the attacks on the nonlinear filter generator are unsuccessful when the fixed key is known, then they will also be unsuccessful when the fixed key is unknown.

Generally, in the case where both the stream key and the fixed key are unknown, the computational complexity of attacks on the generator is equal to the computational complexity of the attack for a known fixed key multiplied by 2^{120} (the number of possible LFSR feedback functions). However, this is not always the case. For example, the simple attack based on linear complexity [MASS 69] does not require knowledge of the LFSR feedback function, or even the length of the LFSR, so the complexity of this attack is unchanged by knowledge of the fixed key. However, all of the more sophisticated attacks outlined in Section 5.2 and 5.3 below require knowledge of this LFSR. Hence, in the case where the LFSR is unknown the complexity of each attack should be multiplied by a factor of 2^{120}.

5.1 Simple Attacks

There are four simple, classical attacks on pseudorandom sequence generators. These are brute force attacks, in which exhaustive search of the keyspace is conducted; attacks based on short period; attacks based on low linear complexity and statistical attacks where the output sequence is sufficiently non-random. Assuming the fixed key is known, for a brute force attack on the TOYOCRYPT-HR1 pseudorandom sequence generator, there are 2^{128}–1 stream keys to be tested. For an attack based on repetition of the period, more than 2^{128}–1 bits of keystream must be observed. Similarly, an attack based on the linear complexity would require a known keystream segment of at least 2^{125} bits, and very likely more, say 2^{128} bits. The statistical analysis described in Section 4.3 identified no weakness which could be used in an attack. Thus the TOYOCRYPT-HR1 keystream generator is resistant to these simple attacks, as the keyspace is too large to permit exhaustive search, the period and linear complexity are too high to be the basis of attacks and subsequences examined displayed white-noise statistics.

5.2 Divide and Conquer Attacks

Divide and conquer attacks are commonly applied to keystream generators with multiple component LFSRs. The attacks work on the components of the generator separately and sequentially solve for the individual subkeys, generally the initial states of the component LFSRs. This style of attack is not commonly used on nonlinear filter generators, as these have a single LFSR. However, as noted in Section 3, the inputs to the function come from distinct sections of the LFSR: X_L, X_R, and x_{127}. This property of the Boolean function f may be exploited in an attack performed in a divide and conquer manner by considering these sections of the LFSR as separate components. A similar approach was used to attack the ORYX cipher [WAGN 98]. The attack is conducted under the assumption that the fixed key (the LFSR feedback function) is known. An outline of the procedure for a divide and conquer attack on the TOYOCRYPT-HR1 keystream generator follows.

5.2.1 Attack procedure

The attack targets X_R. For an assumed initial state of X_R, the values of the terms of f of order greater than two are known, and one factor in each of the quadratic terms is also known. Thus the keystream bit can be equated to a known linear combination of x_{127} and some of the bits of X_L. The identity of these bits is known from a combination of the known quadratic terms in the filter function together with the assumed value of X_R. A branching search process using a linear consistency test [ZENG 90] can be used to determine whether the guessed initial state of X_R was correct.

The search process in the attack procedure is iterative. In each iteration, a value for x_{127} is guessed, a linear equation obtained, and a consistency check made. If the equation is consistent with previous linear equations, the guessed value of x_{127} and the known feedback function are used to step the LFSR. The process is repeated, forming a path of successive consistent guesses. Note that each guess involves choosing one of two possible values for x_{127}, and so marks a branching point on the path. Note also that each time the LFSR is stepped, another bit of X_L is obtained, reducing the number of unknown variables in the subsequent linear equations.

If, in some iteration, a linear equation is obtained which is inconsistent with previous equations, then assume that a guessed value of x_{127} is incorrect. Retrace the path of guesses to the last branching point. Discard all linear equations obtained after this point. Change the guessed value, and begin to trace out another path. If all possible paths result in inconsistencies, then the guessed initial state of X_R was incorrect.

5.2.2 Attack algorithm

Input:The LFSR feedback polynomial; the observed segment of the pseudorandom output sequence of length N, $\{z(t)\}_{t=0}^{N-1}$.

1. Set t=0. Initially, the pool of linear equations is empty.
2. Guess the contents of $X_R(t)$.
3. Guess the contents of $x_{127}(t)$. Note this is a branching point.
4. Substitute values from $X_R(t)$ and $x_{127}(t)$ into f to form a linear equation in terms of bits of X_L Equate to the known keystream bit $z(t)$.
5. Check whether the new linear equation is consistent with all previous equations in pool. If consistent, add to pool. If inconsistent, go to Step 7.
6. Increment t. If $t \le N$, step the LFSR, using the known feedback function and the guessed value of $x_{127}(t)$ and go to Step 3. If $t>N$, go to Step 10.
7. Go back along the path of guesses for $x_{127}(t)$ to the last branching point. Change the value of t to reflect this retracing of the path. Delete from the pool those linear equations obtained after this point. Change the value of $x_{127}(t)$, and note that this is no longer a branching point. Go to Step 4.

8. If all paths are searched without finding a path of consistent guesses equal to the length of the known keystream, then the guessed contents of $X_R(0)$ were obviously wrong. Go to Step 1.
9. Correct guess for X_R forms part of LFSR initial state, use correct path of guessed values of x_{127} to recover the rest of the LFSR initial state.
10. Stop the procedure.

Output: Initial state of LFSR.

5.2.3 Implementation issues for the attack

The attack involves guessing the initial contents of X_R, assuming the guess to be correct, using this guess and the further guessed contents of x_{127} to form a pool of linear equations, and testing whether each new linear equation is consistent with those already in the pool. If inconsistencies arise for a guessed value of x_{127}, we conclude that the guessed value was incorrect.

5.2.3.1 False alarms

A false alarm occurs when a wrong guess is not detected; that is, when the linear equations are consistent, even though the initial guess for X_R is incorrect. Note that the guessed value of X_R and the first guessed value of x_{127} are used to form the first linear equation in the pool. As this is the first equation, no inconsistencies will be detected at this point. However, as the iterative process continues, each guessed value of x_{127} gives an extra linear equation, while removing one variable. That is, for the first iteration, we have one linear equation in (potentially) 63 variables, for the second iteration, we have two linear equations in 62 variables, and so on. If our path of guessed values continues, by the 32nd iteration, there will be 32 linear equations in 32 variables. Of course, it is possible that inconsistencies will have been detected before reaching this point. Certainly it is unlikely that an incorrect initial state will produce a set of consistent linear equations for many iterations past this. The probability of detecting a wrong guess increases with each iteration.

5.2.3.2 Missing the event

Missing the event occurs when the correct initial state is not identified. In this case, if the initial value of X_R, and subsequent path of values for x_{127} are guessed correctly, then all linear equations will be consistent. Thus, the correct initial state will always be identified when tested. The probability of missing the event is zero.

5.2.3.3 Effect of length of known keystream

This is a known plaintext attack. To perform this attack, at least 65 bits of the pseudorandom output sequence must be known. One bit of keystream is required for each guessed value of x_{127} and, after the initial guess of X_R, a path of 65 consecutive correct guesses for x_{127} is required to fill the LFSR. Additional bits can be used to resolve any possible ambiguity.

5.2.4 Attack complexity

This divide and conquer attack requires exhaustive search of the keyspace for X_R, and a linear consistency test for each additional guess of x_{127}. For the correct initial state of X_R, a series of 128 – 63 = 65 correct guesses of x_{127} is needed to complete the recovery of LFSR initial state. For a wrong guess of X_R, inconsistencies in the pool of linear equations should be detected before more than 32 iterations of the algorithm. The complexity of the divide and conquer attack is therefore expected to be $O(2^{95})$. It should be noted that this is only an approximation of the complexity. Due to the magnitude of this complexity no computer simulations to verify this value have been performed. As noted above, there are many uncertainties in the attack model making it difficult to determine the probability of a false alarm.

5.3 Correlation Attacks

Divide and conquer correlation attacks had their origin in a paper by Siegenthaler in 1985 [SIEG 85], in which an attack on a nonlinear combiner generator was presented. The attack is based on a model in which the keystream is viewed as a noisy version of an underlying LFSR sequence, with the noise assumed to be additive and independent of the underlying sequence. The correlation between each of the inputs to a combining function and the output could be exploited, to sequentially recover the initial states of the component LFSRs. The attack required sequential exhaustive search of the initial states of the component LFSRs, so would perform no better than exhaustive search if applied to a nonlinear filter generator.

5.3.1 Fast Correlation Attacks

Fast correlation attacks are correlation attacks that outperform exhaustive search over the initial states of the component LFSRs. Meier and Staffelbach proposed such an attack on the nonlinear combiner generator in 1989 [MeSt 89]. The attack was modified and applied to the nonlinear filter generator by Forre [FORR 90]. However, the complexity of this attack is still exponential in the length of the LFSR. If applied to TOYOCRYPT-HR1, the complexity would be $O(2^{128})$, which is infeasible. Additionally, the success of the attack is dependent on the values of crosscorrelation between the output sequence and the underlying LFSR; for crosscorrelation values which are much less than 75%, as is the case for TOYOCRYPT-HR1, the attack is not successful.

5.3.2 Linear Cryptanalysis

Linear cryptanalysis of stream ciphers was investigated using a linear sequential circuit approximation by Golic [GOLI 94]. This attack was applied to nonlinear filter generators in [SALM 97]. This attack is a more efficient fast correlation attack on the nonlinear filter generator than that described in Section 5.3.1. The attack is based on a model in which the keystream is regarded as a noisy version of some linear transform of the underlying LFSR sequence, with the probability of noise different from one half [GOLI 96]. Hence the keystream sequence satisfies the same linear recurrence as the LFSR sequence, again with probability different from one half. It was shown that fast correlation attacks using iterative probabilistic error correction can be successfully applied to the nonlinear filter generator provided the level of noise is not too close to 0.5, and if enough low weight parity checks can be obtained. The probability of noise depends on the correlation coefficients of the filter function to linear functions, and may be close to 0.5 if the filter function is close to a bent function. The precomputational complexity of this attack is exponential in the number of inputs to the filter function, and the computational complexity is proportional to the length of known keystream and the average number of parity-checks per bit used. For the filter function specified for TOYOCRYPT-HR1, the number of inputs to the filter function makes the attack infeasible, as the precomputational complexity is $O(2^{127})$. Additionally, the probability of noise (as calculated in [EVAL HR1]) is given by $p = \frac{1}{2} - 2^{-64}$. This noise value is so close to 0.5 that unconditional fast correlation attacks will not be successful, regardless of the existence or otherwise of the low-weight parity check polynomials these attacks require.

5.3.3 Conditional Correlation Attacks

The attacks outlined above are unconditional correlation attacks. There exist other conditional correlations in nonlinear filter generators that have been exploited in attacks. In [ANDE 95] an augmented filter function is considered, so that the correlation between blocks of successive input and output bits may be considered. The attack requires precomputation with computational complexity exponential in four times the number of LFSR stages spanned by the inputs to the filter function, that is $O(2^{512})$, and so is infeasible to apply to TOYOCRYPT–HR1.

5.3.4 Inversion Attack

Another attack on nonlinear filter generators is the inversion attack [GOLI 96]. For this attack, the nonlinear filter generator is viewed as a finite input memory combiner, with one input and one output. The input memory space is the number of LFSR stages spanned by the inputs to the filter function. Although not a correlation attack, the inversion attack essentially exploits the correlation to recover the unknown LFSR sequence from an assumed input memory state. The computational complexity of the attack is exponential in the size of the input memory space. For TOYOCRYPT-HR1, the input memory space is 128, and so the computational complexity is $O(2^{128})$. Therefore it is infeasible to apply the inversion attack to TOYOCRYPT–HR1.

5.4 Discussion

In this section of the report, the security of the TOYOCRYPT-HR1 pseudorandom sequence generator has been examined with respect to common attacks on LFSR-based keystream generators. The TOYOCRYPT-HR1 design produces sequences with long period and large linear complexity, so that simple attacks based on these properties are infeasible. Also, the large number of inputs and high nonlinearity of the filter function f provides resistance to correlation attacks, including fast correlation and conditional correlation attacks, and to the inversion attack. These attacks were examined under the assumption that the structure of the generator was known to the cryptanalyst, and that only the stream key was unknown. The attacks were no better than a brute force attack: an exhaustive search of the stream key keyspace.

Of the possible attacks examined, only a divide and conquer attack outperformed the brute force attack. In the case where the LFSR is known the expected complexity of the divide and conquer attack is $O(2^{95})$, a considerable reduction from the 2^{128} possible stream keys. In the case where the LFSR is not known this complexity should be multiplied by 2^{120}. Note that it may be possible to slightly reduce the complexity of this attack by using the repeated common differences in quadratic terms described in Section 3.3. The amount of known plaintext required for this attack is extremely small, only 65 bits.

The divide and conquer attack is possible only because of a feature of the filter function f: all of the inputs to the higher order terms of the function come from X_R, a 63-bit section of the LFSR. This attack can be made more difficult by a minor adjustment to f. By taking the inputs to these higher order terms from stages spanning the entire LFSR, the attack becomes infeasible. This minor adjustment will retain the nonlinearity and balance of f provided that the constraints of the bent function construction are satisfied, (see [ROTH 76]).

6 Implementation

As discussed above, the structure of TOYOCRYPT-HR1 is relatively standard for a stream cipher. It uses standard components, namely a linear feedback shift register (LFSR) and a nonlinear Boolean function. The Galois representation of the LFSR is used to enhance the efficiency of its implementation. This is important since the LFSR will generally be the bottleneck in a stream cipher implementation (as it is in this case).

6.1 Software

The TOYOCRYPT-HR1 specification [SPEC HR1] describes two software implementations – a serial implementation and a parallel implementation. The parallel implementation is significantly more efficient but implements a slightly different mode of operation in that it generates a number of independent sequences in parallel rather than a single keystream sequence.

The evaluators agree with the analysis (in [SPEC HR1], Section 6.3) of the number of operations required in the computation of f (420 operations) and the number of operations required to update the LFSR (639 operations). The total number of operations given for the serial (650) and parallel (1060) implementations for the generation of one and *Len* bits (where *Len* is the word size of processor), respectively, also appear to be correct.

The serial mode of TOYOCRYPT-HR1 was implemented (but not optimised) in C on an Intel Pentium processor by the review team and was found to run at speeds similar to those reported in the Section 6.2 of [EVAL HR1]. The results of the serial implementation in software show that the cipher will encrypt at only 9.56kbps on a RISC processor at 20MHz. The evaluators agree with the developers comment that an implementation in assembly language would at most improve this speed by a factor of 2 to 3. The speed of the serial implementation in software appears to be far below what is considered acceptable by today's standards.

The parallel implementation generates up to *Len* independent keystream sequences in parallel. Each of the independent keystreams requires a separate initial state S and feedback polynomial C. The parallel implementation would run approximately 1.6 times slower than the serial implementation but could compute up to *Len* keystream sequences in parallel. On a 32-bit RISC processor running at 20MHz the parallel implementation will encrypt 32 independent keystreams at an overall rate of 174kbps. This speed would be unacceptable for high bandwidth applications.

The bitslice technique presented in the parallel implementation is not specific to the TOYOCRYPT-HR1 algorithm and could be applied to any similar stream cipher which is based on bit operations.

The size of the program code for the TOYOCRYPT-HR1 algorithm is extremely small, making it suitable for implementation in devices with memory restrictions such as smart cards.

6.2 Hardware

The TOYOCRYPT-HR1 self evaluation report [EVAL HR1] describes the results of simulations of hardware implementations on two different platforms. The first, an

FPGA implementation, gives encryption rates of between 81 Mbps (worst case, not optimised for speed) and 418 Mbps (best case, optimised for speed) depending upon the circuit size and for a range of voltages. The second platform tested was a gate array processor with the reported data rates ranging from 223 Mbps (worst case) to 790 Mbps (best case).

It was not feasible for the evaluators to verify these results. As expected, hardware implementations of stream ciphers are much faster than software implementations. The encryption rates given for hardware implementations of the TOYOCRYPT-HR1 algorithm are adequate for most applications. For applications requiring very high bandwidths (> 1Gbps) this algorithm would not be suitable.

6.3 Key Generation

The TOYOCRYPT-HR1 key includes a 128-bit value, *C*, which represents the coefficients of a monic primitive polynomial of degree 128. The key setup times quoted in the TOYOCRYPT-HR1 documentation (Section 6.2 of [EVAL HR1]) do not include the non-negligible time required to compute this polynomial. Algorithm 4.6 of [SPEC HR1] describes an algorithm for generating *C*. This algorithm is identical to previously known techniques [MOV 97] and requires first finding a monic irreducible polynomial before testing that it is primitive. As the self evaluation suggests (Section 3 of [EVAL HR1]), there are precisely:

$$\frac{\phi(2^m - 1)}{m} \approx 2^{119.9976}$$

monic primitive polynomials of degree m=128. This agrees with the TOYOCRYPT-HR1 designers' evaluation that the effective key space is reduced from 256 bits to 248 bits (128 bits from *S* and 120 bits from *C*).

The proportion of random monic polynomials of degree m that are irreducible is $1/m$. And the complexity of the test for irreducibility is $O((\lg m)(\lg 2)m^2)$ [MOV 97]. For m=128, the proportion of irreducible polynomials that are primitive is approximately:

$$\frac{\phi(2^m - 1)}{2^m} \approx 0.4992$$

Thus, on average, one would expect to perform 64 irreducibility tests and two tests of primitiveness to find a suitable *C*. The high complexity of the tests for irreducibility and primitiveness suggest that the time required to find a suitable polynomial may be considerable.

The evaluators performed a software implementation (not optimised for speed) of the algorithm for generating random monic primitive polynomials of degree 128 which took a number of minutes, on the average, to complete. It is expected that a highly optimised implementation would take several seconds to generate a suitable polynomial. This is an important consideration when re-keying of the algorithm is required (see Section 3.3 of this report).

7 Conclusion

The TOYOCRYPT-HR1 pseudorandom number generator has been designed using a LFSR together with a nonlinear Boolean function. The evaluators agree with the report [EVAL HS1] that the standard keystream properties of large period, large linear complexity and white-noise statistics are satisfied by this generator. However, the evaluators have identified three major areas of concern.

Firstly, we have outlined a divide and conquer attack which requires only a small amount of known keystream and has complexity approximately $O(2^{215})$ if the LFSR is unknown and $O(2^{95})$ if known. In comparison an exhaustive key search has complexity of $O(2^{248})$ and $O(2^{128})$ respectively for these two cases. It is possible to prevent this attack by a minor adjustment to nonlinear filter function (see Section 5.4).

The second area of major concern relates to the speed of TOYOCRYPT-HR1 in software. The performance of TOYOCRYPT-HR1 in software is very slow. The designers of algorithm have recognised this problem and have included a parallel mode of operation. The evaluators agree that this process will speed up operation. However it should be noted that this will also require a much larger key size.

The third major area of concern relates to generation of primitive polynomials as part of the key rather than using a fixed LFSR. As we have noted this process is extremely slow and may not be suitable for many cryptographic applications such as protocols involving Diffie-Hellman key agreement. Due to this it may be necessary to use a fixed LFSR in many applications, reducing the effective key size to 128 bits.

The second and third area of concern relate to implementation issues. There seems to be no simple way to overcome these problems.

8 References

[ANDE 95]
R. J. Anderson, Searching for the optimum correlation attack, In *Fast Software Encryption – Leuven'94,* Volume 1008 of *Lecture Notes in Computer Science,* pages 137-143. Springer-Verlag, 1995.

[BS 90]
E. Biham and A. Shamir, Differential cryptanalysis of DES-like cryptosystems, In *Advances in Cryptology – CRYPT0 '90,* Volume 537 of *Lecture Notes in Computer Science,* pages 2-21. Springer-Verlag, 1991.

[EVAL HR1]
Self Evaluation Report TOYOCRYPT-HR1. Technical Document, October 2000.

[EVAL HS1]
Self Evaluation Report TOYOCRYPT-HS1. Technical Document, October 2000.

[FORR 90]
R. Forre, A fast correlation Attack on nonlinearly feedforward filtered shift-register sequences. In *Advances in Cryptology – EUROCRYPT '89*, Volume 434 of *Lecture Notes in Computer Science*, pages 586-595. Springer-Verlag, 1990.

[GOLI 94]
J. Golic, Linear cryptanalysis of stream ciphers, In *Fast Software Encryption – Leuven'94*, Volume 1008 of *Lecture Notes in Computer Science*, pages 154-169. Springer-Verlag, 1995.

[GOLI 96]
J. Dj. Golic, On the security of nonlinear filter generators, In *Fast Software Encryption – Cambridge'96*, Volume 1039 of *Lecture Notes in Computer Science*, pages 173-188. Springer-Verlag, 1996.

[MASS 69]
J. L. Massey, Shift Register Synthesis and B.C.H. decoding, *IEEE Trans. Inform. Theory*, IT-15:122-127, January 1969.

[MeSt 89]
W. Meier and O. Staffelbach, Fast correlation attacks on certain stream ciphers, *Journal of Cryptology*, 1(3):159-167, 1989.

[MOV 97]
A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[ROTH 76]
O. S. Rothaus, On bent functions, *Journal of Combinatorial Theory(A)*, Volume 20, pages 300-305, 1976

[SALM 97]
M. Salmasizadeh, L. Simpson, J. Dj. Golic and E. Dawson, Fast correlation attacks and multiple linear approximations, In *Information Security and Privacy – ACISP'97*, Volume 1270 of *Lecture Notes in Computer Science*, pages 228-239. Springer-Verlag, 1997.

[SIEG 85]
T. Siegenthaler, Decrypting a class of stream ciphers using ciphertext only, *IEEE Trans. Comput.*, C-34:81-85, January 1985.

[SPEC HR1]
Cryptographic Techniques Specifications TOYOCRYPT-HR1. Technical Document, October 2000.

[SPEC HS1]
Cryptographic Techniques Specifications TOYOCRYPT-HS1. Technical Document, October 2000.

[WAGN 98]
D. Wagner, L. Simpson, E. Dawson, J. Kelsey, W. Millan and B. Schneier, Cryptanalysis of ORYX, In *Workshop on Selected Areas in Cryptography (SAC'98)*, Volume 1556 of *Lecture Notes in Computer Science, pages 296-305. Springer-Verlag, 1998.*

[ZENG 90]
K. C. Zeng, C. H. Yang and T. R. N. Rao, On the linear consistency test (LCT) in cryptanalysis with applications, In *Advances in Cryptology – CRYPT0 '89*, Volume 434 of *Lecture Notes in Computer Science*, pages 164-174. Springer-Verlag, 1990.

9 Appendices

Appendix A. Description of Statistical Tests

This appendix gives a mathematical description of the statistical tests used.

A.1 Mathematical Description of Stream Cipher Tests

This section contains a mathematical description of each of the tests. 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.

A.1.1 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:

$$n = 50$$

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

$$n_1 = 21$$

$$\hat{p} = 0.42$$

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

$$p = 0.2579$$

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.

A.1.2 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) :

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

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) :

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

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.

A.1.3 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 $\mathrm{U[t]} = \mathrm{n} \times \mathrm{S[t]} - \mathrm{t} \times \mathrm{S[n]}$ 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.

A.1.4 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 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 :

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

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 repetitions tests:

$b = 18$ (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.

A.1.5 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

$k = \lfloor \log_2 \frac{n+1}{5} - 1 \rfloor$.

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 ≥ 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 ≥ k

The variables:

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

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

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

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

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 $j = 1,...,2k$ using above formulae.

4. 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:

$n = 50$

$\text{total runs} = 31$

$n_1 = 21$

$k = \lfloor \log_2 \tfrac{n+1}{5} - 1 \rfloor = \lfloor \log_2 \tfrac{51}{5} - 1 \rfloor = 2$

$s_{01} = 10,\ s_{02} = 5,\ s_{11} = 13,\ s_{12} = 3$

$$u_1 = x_1 = \frac{13 - 50(\frac{1}{2})^3}{\sqrt{50}} = 0.9545941546018$$

$$u_2 = x_2 = \frac{3 - 50(\frac{1}{2})^3}{\sqrt{50}} = -0.4596194077713$$

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

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

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

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

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

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

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

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

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

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

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

$\sigma^{11} = 6\tfrac{2}{3}$, $\sigma^{12} = -5\tfrac{1}{3}$, $\sigma^{13} = -4$, $\sigma^{14} = -3\tfrac{1}{3}$, $\sigma^{21} = -5\tfrac{1}{3}$, $\sigma^{22} = -5\tfrac{1}{3}$, $\sigma^{23} = 0$, $\sigma^{24} = -5\tfrac{1}{3}$,
$\sigma^{31} = -4$, $\sigma^{32} = 0$, $\sigma^{33} = 12$, $\sigma^{34} = 2$, $\sigma^{41} = -3\tfrac{1}{3}$, $\sigma^{42} = -5\tfrac{1}{3}$, $\sigma^{43} = 2$, $\sigma^{44} = -\tfrac{1}{3}$.

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*
>
> .

A.1.6 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

$\quad i = i + \left\lfloor \frac{\log(i-1)}{\log(2)} \right\rfloor + 2;$

$\quad c = c + 1;$

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:

$n = 50$

$c(s) = 10$

Threshold value $= 8.859191$

Expected value $= 13$

Interpretation:

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

A.1.7 Linear Complexity Test

A.1.7.1 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:

01011001010100100111100000110111001100011101011111101101

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 $2^{\mathrm{L}} - 1$ and a linear complexity of L. An m-sequence has the *best* possible distribution of zeros and ones for a sequence of period $2^{\mathrm{L}} - 1$. In this fashion an m-sequence appears to be *statistically* random in terms of tests A.1.1 to A.1.6. 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.

A.1.7.2 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:

A.1.7.3 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]. 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.

A.1.7.4 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

$$n = 50$$

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

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

$$L(s) = 25$$

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

$$p = 0.5$$

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

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

$e_1 = 7.5$ 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 $f_{2+} = 4$.

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

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

$$p = 0.0707$$

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.

A.2 Mathematical Description of Key Generator Tests

The security of cryptographic devices used in large telecommunication networks such as the Internet depends, to a large extent, on the secrecy of keys (passwords) that initialise the process, the strength of the encryption algorithms used, and the security architectures incorporated.
The tests designed for measuring the independence of plaintext and ciphertext in block ciphers are adapted to testing the strength of key generators for symmetric ciphers. These tests are applied under the black-box approach such that no knowledge of the algorithm is required, only the key size is required. In applying the tests of randomness to a set of k-bit keys output from a key generator for any one seed, the data is treated as a stream of binary k-bit blocks.

A.2.1 Frequency Test

Let f_i denote the number of n-bit blocks whose count of ones is i, from the set of S n-bit blocks to be examined for randomness. There are altogether ${}^nC_i$ possible binary blocks of length n and *weight* i, where *weight* refers to the sum of the bits in the block. It should be noted that ${}^nC_i = \frac{n!}{(n-i)!\,i!}$. In the case where the S n-bit blocks are random, the expected number of binary blocks from this set of weight i, e_i, is given by $e_i = \frac{{}^nC_i S}{2^n}$ [DAWS 91]. If the S blocks are random then the distribution of the f_i values should be approximately the same as that of the hypothesised e_i values.
The values for f_i and e_i can be compared using the chi-squared, χ^2, goodness-of-fit test: $\chi^2 = \sum_k \frac{(f_i - e_i)^2}{e_i}$ where n should be taken large enough so that each $e_i \geq 5$.
Otherwise the smaller e_i values, (for values of i near 0 and values of i near n), should be grouped so that no e_i value is less than 5. The f_i values are grouped correspondingly. Assume there are k resulting values of $e_i \geq 5$. This results in the number of degrees of freedom being one less than the resulting number of e_i values obtained (k-1).
The significance probability, p, for the χ^2 distribution is calculated. A small significance probability indicates a significant result. For large values of n a highly significant result ($p < 0.001$) indicates a possible weakness in the cipher.
Note that if the plaintext contains a large number of repetitions of any block(s), then this should yield a significant result for this test. It is recommended that no further tests be carried out on this sample, as their results may be affected by this distribution of ones.

A.2.2 Binary Derivative Test

The binary derivative is a new bit stream formed by the modulo two addition of successive bits in the stream.

E.g. The binary derivative of `01101000100111`
is `1011100110100`

In the example:

a 01 in the block becomes a 1 in the first derivative,
a 10 in the block becomes a 1 in the first derivative,

a 00 in the block becomes a 0 in the first derivative,
a 11 in the block becomes a 0 in the first derivative.

The frequency test is applied to the sample of binary derivative blocks.

Parameters needed: S = number of blocks
n-1 = length of binary derivative block (in bits)

By taking the binary derivative of each of the S blocks to be examined, a set of S binary blocks of length *n*-1 are formed. Let f_i' denote the number of these derivative blocks of *hamming weight* i, where *hamming weight* refers to the sum of the bit values in the block.

There are altogether ${}^{n-1}C_i$ possible binary blocks of length n-1 and weight i. In the case where the S binary blocks are random, the expected number of binary blocks from this sample of weight i, e_i', is given by $e_i = \frac{{}^{n-1}C_i S}{2^{n-1}}$ [DAWS 91].

If the S binary derivative blocks are random then the distribution of the f_i' values should be approximately the same as that of the hypothesised e_i' values. The values for f_i' and e_i' can be compared using the chi-squared, χ^2, goodness-of-fit test:

$\chi^2 = \sum_k \frac{(f_i' - e_i')^2}{e_i'}$ where n should be taken large enough so that each $e_i' \geq 5$.

A similar interpretation to the frequency test is applied.

A.2.3 Subblock Test

Subblocks of length b bits are chosen from each block. The user has the option to select the bit positions, or they may be randomly selected.

There are four testing procedures for subblock lengths: b = 1, 2, 3 to 16, and greater than 16.

b = 1: The key blocks will be tested for randomness in each of the n bit positions. The i^{th} position, $1 \leq i \leq n$, in each of the S blocks gives a stream of S bits to which a *frequency* test (as used for the stream cipher – see Section A.1.1) is applied. This determines weaknesses in any single bit position.

Since the frequency test is highly sensitive to small deviations from the mean, the n results will not be combined to give an overall statistic for this test. A summary of results will be given, so that any problems with individual bit positions may be detected.

For significance probabilities less than 0.001 in this test, it is highly unlikely that further subblock tests for *b* > 1 will give significant results when including any one of these bits.

b = 2: For $i \neq j$, $1 \leq i \leq n$ and $1 \leq j \leq n$, a *uniformity* test with b = 2 is applied to each of the ${}^nC_2$ pairs of values in position i and j for each block. The test statistic used is

$\chi^2 = \frac{4}{S}\sum_{i=0}^{3} f_i^2 - S$, which follows a chi-squared distribution with three degrees of freedom. This determines whether ciphertext bits i and j are independent or if a dependence exists between them.

Since each bit position is used in more than one of the ${}^nC_2$ tests applied, the chi-squared statistics obtained are not independent, and so cannot be combined into one test statistic. A summary of significance probabilities will be given.

b ≥ 3: The tests applied to subblocks of length greater than three are the same as for the stream cipher subblock tests (see Section A.1.4), i.e. if b ≤ 16 then a test of uniformity is applied, whereas if b > 16 the repetition test is applied.

A.2.4 Block Entropy

The number of independent bits in the block is calculated, using the application of the *repetition test* (see Section A.1.4), to estimate the entropy of the block.

Method of calculating entropy:
Using the results of the *repetition test*, a 99% probability interval for the mean number of block (or subblock) repetitions, λ, that the cipher could generate in a sample of size S where $\lambda = r \pm 2.5758\sqrt{r}$, is given by applying a normal approximation to the Poisson distribution. This is used to find a 99% probability interval for the size of the population, N, of blocks (or subblocks) that could be generated by such a cipher, using $N = \frac{S^2}{2\lambda}$ [DAWS 98]. Since the total number of binary patterns for a block/subblock of B bits is 2^B, then the number of independent bits, H, is estimated from a table of values $N = 2^H$. The values of H (or in the case of subblocks, H_i) corresponding to the end points of the probability interval for N give a probabilistic interval for the entropy of the block (or subblock).
If the number of repetitions is found to be greater than 48 (approximately three standard deviations above the mean) for a sample of less than $2^{\frac{B}{2}+3}$ blocks (B = block/subblock size) then the cipher is considered to generate non-random data. A message will be given in such cases. The results will give the estimate of block entropy. If subblocks are used, then a table of entropy estimates for the subblocks will also be included. If the entropy is less than the block size, then the reduction indicates the number of dependent bits in the block.
If the entropy of the whole block was calculated then a 99% probability interval for the true *entropy of the block*, H, is given. If the block has been divided into k subblocks, then the *block entropy*, H, is estimated by the bounds
$\mathrm{Max}(H_i) \le H \le \sum_{i=1}^{k} H_i$ where the lower bound is the maximum subblock entropy and holds with a probability of 0.995, and the upper bound is the sum of the k subblock entropies and holds with a probability of 0.995^k.
If the sample of blocks gives at least 20 repetitions, then the *block entropy* is estimated. In the case where the number of blocks is greater than $2^{\frac{n}{2}+3}$ then the estimate for block entropy will be given using no more than 2^{n-3} blocks. If the number of repetitions is less than 20, then a whole block estimate will not be given, and instead, a bound for the entropy is calculated by combining entropy estimates from smaller subblocks. In this case the block is divided into subblocks of length B, where the maximum length, B, of the subblocks is determined from the number of blocks to be tested, S, such that $S = 2^{\frac{B}{2}+3}$.

Any remaining bits, d, are tested using a sample of $s = 2^{\frac{d}{2}+3}$ subblocks of d bits (d < B) from these positions in each block. If $d \le 16$ then the requirements of the repetition test are not satisfied. In this case the entropy of the subblock is taken as d, the subblock size. The entropy is estimated for each subblock, as long as the number of subblock repetitions is at least 20. If the number of repetitions is less than 20 then the entropy estimate cannot be assumed to be any less than the length of the subblock. These estimates are then added to give an upper bound for the whole block entropy.

The best application of this estimation is applied to samples of approximately $2^{\frac{B}{2}+3}$ sublocks of length B bits, where B > 16 and for which the expected number of repetitions is 32. The following table gives the entropy estimate for a sample of one million blocks, when the number of repetitions is 30, 50 or 100. It should be noted that this is independent of the block size used.

Repetitions	Entropy estimate
30	34
50	33
100	35

In the case when 30 repetitions occur, the minimum sample size, S, required to obtain a 0.99 probability interval for the block entropy, H, that includes the actual block size, is given in the table below. If 30 repetitions occurred in a smaller sample, or more than 30 repetitions occurred in a sample of the same size, then the block entropy would be lowered.

Actual key size	0.99 probability interval for H	Sample size required, S
32	31.5 – 32	3.7×10^5
40	38.5 – 40	6.0×10^6
56	54.5 – 56	1.5×10^9
64	62.5 – 64	2.4×10^{10}
80	78.5 – 80	6.2×10^{12}
128	126.5 – 128	1.04×10^{20}

A.3 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.

[DAWS 98] E.P. Dawson and H.M. Gustafson, "A Method for measuring Entropy of Symmetric Cipher Key Generators", **Computers and Security**, Vol. 17 No. 2, pp 177 - 184, 1998.

[DIFF 79] W. Diffie and M.E. Hellman, "Privacy and Authentication: An Introduction to Cryptography", **Proceedings of the IEEE**, Vol. 67, No. 3, March 1979, pp 397-427.

[FOLK 84] L.J. Folks, "Combination of Independent Tests", **Handbook of Statistics**, Vol. 4, Elsevier Science Publishers, 1984, pp 113-121.

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

[KOLC 66] V.F. Kolchin, "The Speed of Convergence to Limit Distributions in The Classical Ball Problem", **Theory of Probability and its Applications**, 11, 1966, 128-140.

[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.

[MAUR 92] U.M. Maurer, "A Universal Statistical Test for Random Bit Generators", **Journal of Cryptology**, 1992, 5, pp 89-105.

[MARS 84] G. Marsaglia, "A Current View of random Number Generators", Proceedings of the Sixteenth Symposium on the Interface, **Computer Science and Statistics**, Editor L. Billard, Elsevier Science Publishers, 1984. pp3 - 10.

[MASS 69] J.L. Massey, "Shift Register Sequences and BGH Decoding", **IEEE Trans. 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.

[STEP 86] M.A. Stephens and R.B. D'Agostino, “Tests based on EDF Statistics”, **Goodness of Fit Techniques**, **in Statistics, Textbooks and Monographs**; Vol. 68, Marcel Dekker Inc., 1986, pp 97-193.

[WEBS 86] A.E. Webster and S.E. Tavares, “On the Design of S-Boxes”, **Advances in Cryptology: Crypto' 85**, Springer-Verlag, 1986, pp. 523-530.

Appendix B.

Algebraic Normal form of TOYOCRYPT-HR1 Boolean function *f*.

$$
\begin{aligned}
f(x_0, x_1, \ldots, x_{125}, x_{127}) = {} & x_{127} \\
& \oplus x_{24}x_{63} \oplus x_{41}x_{64} \oplus x_{27}x_{65} \oplus x_{32}x_{66} \oplus x_{35}x_{67} \oplus x_{50}x_{68} \\
& \oplus x_{8}x_{69} \oplus x_{18}x_{70} \oplus x_{1}x_{71} \oplus x_{36}x_{72} \oplus x_{53}x_{73} \oplus x_{26}x_{74} \\
& \oplus x_{3}x_{75} \oplus x_{7}x_{76} \oplus x_{11}x_{77} \oplus x_{6}x_{78} \oplus x_{62}x_{79} \oplus x_{37}x_{80} \\
& \oplus x_{31}x_{81} \oplus x_{12}x_{82} \oplus x_{9}x_{83} \oplus x_{34}x_{84} \oplus x_{51}x_{85} \oplus x_{61}x_{86} \\
& \oplus x_{25}x_{87} \oplus x_{23}x_{88} \oplus x_{45}x_{89} \oplus x_{14}x_{90} \oplus x_{0}x_{91} \oplus x_{20}x_{92} \\
& \oplus x_{46}x_{93} \oplus x_{38}x_{94} \oplus x_{40}x_{95} \oplus x_{13}x_{96} \oplus x_{28}x_{97} \oplus x_{2}x_{98} \\
& \oplus x_{49}x_{99} \oplus x_{54}x_{100} \oplus x_{5}x_{101} \oplus x_{60}x_{102} \oplus x_{47}x_{103} \oplus x_{4}x_{104} \\
& \oplus x_{16}x_{105} \oplus x_{52}x_{106} \oplus x_{59}x_{107} \oplus x_{55}x_{108} \oplus x_{10}x_{109} \oplus x_{57}x_{110} \\
& \oplus x_{22}x_{111} \oplus x_{15}x_{112} \oplus x_{56}x_{113} \oplus x_{48}x_{114} \oplus x_{29}x_{115} \oplus x_{44}x_{116} \\
& \oplus x_{39}x_{117} \oplus x_{58}x_{118} \oplus x_{17}x_{119} \oplus x_{43}x_{120} \oplus x_{19}x_{121} \oplus x_{21}x_{122} \\
& \oplus x_{42}x_{123} \oplus x_{33}x_{124} \oplus x_{30}x_{125} \\
& \oplus x_{10}x_{23}x_{32}x_{42} \\
& \oplus x_{1}x_{2}x_{9}x_{12}x_{18}x_{20}x_{23}x_{25}x_{26}x_{28}x_{33}x_{38}x_{41}x_{42}x_{51}x_{53}x_{59} \\
& \oplus x_{0}x_{1} \ldots x_{62}
\end{aligned}
$$

Appendix C.

Sorted input differences for quadratic terms – TOYOCRYPT-HR1

Index for first factor	Index for second factor	Difference
62	79	17
50	68	18
53	73	20
41	64	23
61	86	25
35	67	32
32	**66**	**34**
51	**85**	**34**
36	72	36
27	65	38
24	63	39
60	102	42
37	80	43
45	89	44
54	100	46
46	93	47
26	**74**	**48**
59	**107**	**48**
31	**81**	**50**
34	**84**	**50**
49	**99**	**50**
18	70	52
55	**108**	**53**
57	**110**	**53**
52	106	54
40	95	55
38	**94**	**56**
47	**103**	**56**
56	113	57
58	118	60
8	69	61
25	87	62
23	88	65
11	**77**	**66**
48	**114**	**66**
7	**76**	**69**
28	**97**	**69**
1	**71**	**70**
12	**82**	**70**
3	**75**	**72**
6	**78**	**72**
20	**92**	**72**
44	**116**	**72**
9	83	74
14	90	76

43	120	77
39	117	78
42	123	81
13	96	83
29	115	86
16	**105**	**89**
22	**111**	**89**
0	**91**	**91**
33	**124**	**91**
30	125	95
2	**98**	**96**
5	**101**	**96**
15	112	97
10	109	99
4	104	100
21	122	101
17	**119**	**102**
19	**121**	**102**

Bold type has been used for table entries to emphasise multiple quadratic terms with a common difference.

Appendix D.

Results of Linear Complexity Tests on TOYOCRYPT-HR1

Length of output stream = 1,000,000 bits.
Number of streams = 10

Test	**Key1**	**Key2**	**Key3**	**Key4**	**Key5**	**Key6**	**Key7**	**Key8**	**Key9**	**Keya**
Linear Complexity	499998	499998	500002	500001	500000	500000	499999	500000	500000	500000
LC p-value	.0261	.0261	.9739	.8341	.5000	.5000	.1659	.5000	.5000	.5000
LC - Jumps	.2623	.5113	.2965	.1926	.4050	.3429	.0812	.5214	.2415	.8261
LC - Jump Size	.7099	.6622	.3050	.6306	.8919	.6059	.1111	.7357	.0931	.0269

Appendix E.

<u>Results of CRYPT-X Tests on TOYOCRYPT-HR1</u>

Length of output stream = 10,000,000 bits.
Length of output stream = 1,000,000 bits. (Sequence Complexity Test)
Number of streams = 10

Test	**Key1**	**Key2**	**Key3**	**Key4**	**Key5**	**Key6**	**Key7**	**Key8**	**Key9**	**Keya**
Frequency	.6561	.6557	.6557	.6557	.6552	.6557	.1705	.6833	.5858	.0163
Binary Derivative (1)	.2094	.2092	.2090	.2090	.2090	.2088	.1675	.8527	.8064	.1070
Binary Derivative (2)	.1034	.1033	.1033	.1033	.1033	.1034	.1970	.9728	.9496	.4803
Change Point	.5793	.5994	.5790	.5787	.5787	.5787	.0833	.2228	.0930	.7717
Subblock b = 2	.3439	.9706	.3434	.9706	.3429	.9705	.3290	.1736	.6109	.0971
Subblock b = 3	.0185	.0604	.0826	.0185	.0603	.0826	.0300	.2895	.7957	.1030
Subblock b = 4	.6667	.2959	.1012	.2760	.6656	.2955	.7999	.1191	.1647	.0772
Subblock b = 5	.7335	.4502	.4864	.1967	.4876	.7332	.3051	.3836	.4255	.3310
Subblock b = 6	.0859	.0020	.0412	.0978	.5439	.7844	.4027	.8225	.6383	.4330
Subblock b = 7	.5260	.1514	.3687	.9734	.4454	.4265	.8331	.3242	.8283	.5846
Subblock b = 8	.4062	.4001	.1335	.1879	.1365	.0177	.1423	.2726	.3422	.0840
Subblock b = 9	.6126	.1129	.0659	.0882	.3296	.5648	.7061	.0562	.9982	.8099
Subblock b = 10	.6075	.0945	.2173	.2728	.3865	.5614	.3586	.6457	.0902	.0264
Subblock b = 11	.4800	.2656	.1368	.4871	.3414	.2927	.4013	.0724	.3623	.1006
Subblock b = 12	.0358	.3106	.2596	.0389	.7884	.0069	.6168	.2688	.6880	.9286
Subblock b = 13	.2134	.8827	.6305	.1608	.3686	.9117	.7010	.4549	.6942	.8530
Subblock b = 14	.6850	.6993	.2170	.7144	.8451	.9438	.7418	.2320	.4656	.9309
Subblock b = 15	.4584	.6548	.6577	.2402	.0536	.0607	.0451	.9187	.3087	.4361
Subblock b = 16	.0252	.0083	.0007	.1570	.7279	.5014	.1228	.6901	.7545	.3233
Subblock b = 17	.0047	.2159	.5959	.7237	.4795	.4795	.1116	.2888	.1116	.2888
Subblock b = 18	.4795	.4795	.4795	.8597	.2888	.5959	.8597	.1116	.3768	.0771
Subblock b = 19	.8597	.5959	1.000	.3768	.3768	.0518	.0518	.5959	.8597	.0339
Subblock b = 20	.3768	.4795	.5959	.3768	.4795	.8597	.8597	.2159	.8597	.0216
Subblock b = 21	.5959	.4795	.2888	.4795	.2159	.2888	.2159	.7237	.1573	.3768
Subblock b = 22	1.000	.1573	.4795	.7237	.7237	1.000	1.000	.0771	.1573	.2888
Subblock b = 23	.2888	.1116	.1116	.1573	.3768	.4795	.2888	.0399	.2159	.7237
Subblock b = 24	1.000	.3768	.0518	.3768	.8597	.8597	.2888	.8597	.4795	.8597
Subblock b = 25	.4795	.1116	.3768	.2888	.1573	.5959	.2159	.4795	.4795	.8597
Subblock b = 26	.1116	.7237	.0771	.5959	.8597	.5959	.0027	.4795	1.000	.1573
Subblock b = 27	.4795	.8597	.1573	.7237	.8597	.5959	1.000	1.000	.3768	.4795
Subblock b = 28	.4795	.4795	.1573	.8597	.4795	.5959	.4795	.5959	1.000	1.000
Subblock b = 29	.4795	.5959	.8597	.2888	.0339	.0771	.3768	.1116	.7237	.8597
Subblock b = 30	.1116	.0771	.3768	.7237	1.000	.2888	.8597	.3768	.2159	.1116
Runs Distribution	.8082	.8084	.8083	.8083	.8084	.8086	.6265	.1958	.8627	.7661
Longest Run Length	25	25	25	25	25	25	21	23	24	23
Sequence Complexity	50795	50795	50794	50795	50794	50795	50829	50771	50799	50742

Sequence Complexity: Threshold value = 50,441   Mean value = 50172

Session Keys Used (HEX):
1: 00000000000000000000000000000001
2: 00000000000000000000000000000002
3: 00000000000000000000000000000004
4: 00000000000000000000000000000008
5: 00000000000000000000000000000010
6: 00000000000000000000000000000020
7: 38b3c50fd5b1bccc49e786ebf28f5a51
8: 04b81a3a854e38b6775cbf4caeb4f1e5
9: f2a09ae954cda165661efc765aeeb0c3
a: 9f9e5ded6e10cfa557c1fe0ed02f1f3f

Fixed key (in HEX): ecb91d4ce0badb56023ca06869e92a2f

Appendix F.

Results of Entropy Tests

Distribution of Subblocks

Length of output stream = 1 Gigabyte
Number of streams = 1

Subblock size	p-value	
	Ones	Runs
40	.1772	.2998
48	.3579	.1554
56	.0300	.9347
64	.2233	.7193
128	.4774	.9479
256	.0700	.7785
512	.5697	.2228
1024	.9494	.5528