International Journal of Cryptology Research 1(2): 129-148 (2009)

LILI to Dragon:
from Bit-Based to Word-Based Stream Ciphers

Ed Dawson and Leonie Simpson
Information Security Institute,
Queensland University of Technology, Australia
E-mail: e.dawson@qut.edu.au;lr.simpson@qut.edu.au

ABSTRACT

Stream ciphers are the standard form of encryption for most applications in secure communications. Recently, many existing stream cipher proposals have been shown to be insecure. Given the wide usage, there have been several attempts at establishing national or global standards for stream ciphers. In 2000, there was a call for stream cipher proposals for use in Europe: the NESSIE project. No suitable candidates were found during this project, and in 2005 the eSTREAM project is again attempting to find stream cipher algorithms suitable for widespread adoption. In this paper we describe and discuss the LILI and Dragon ciphers, submissions to NESSIE and eSTREAM respectively. The security of each cipher is examined. It is shown that both LILI-II and Dragon ciphers are efficient and secure keystream generators for stream ciphers.

INTRODUCTION

A stream cipher is an encryption algorithm which breaks a plaintext message into successive characters and encrypts each character under some time-varying function of the key. Thus different occurrences of the same plaintext character may encrypt to different ciphertext characters. Stream ciphers may be either synchronous or self synchronous. For synchronous stream ciphers, the keystream is generated independently of the plaintext message and the ciphertext, whereas for self-synchronous stream ciphers, the keystream is generated as a function of the key and a fixed number of ciphertext characters.

The plaintext message characters used for stream ciphers operating on electronic data are usually binary digits (bits) or *n*-bit words, as are the ciphertext and keystream characters. The most common form of synchronous stream cipher is the binary additive stream cipher, where a binary keystream (a sequence of bits) is generated independently of the plaintext and the ciphertext. The keystream is combined with the plaintext or ciphertext using bitwise modulo two addition of the two streams, to form ciphertext or plaintext, respectively. Let $p(t), c(t)$ and $z(t)$ represent, respectively, the plaintext, ciphertext and keystream characters at time t, for $t \geq 0$. Then

$c(t) = p(t) \oplus z(t)$, where $\oplus$ denotes addition modulo 2. Similarly, $p(t) = c(t) \oplus z(t)$. A major advantage of this system is that the same device can be used to perform both encryption and decryption.

Note that if some segment of the plaintext is known, then by combining it with the corresponding ciphertext, a segment of the keystream is revealed, since $z(t) = p(t) \oplus c(t)$. The cipher cannot be considered secure if it is possible to predict the entire keystream from a shorter known segment. Thus the keystream generator is the critical security component of the binary additive synchronous stream cipher. Security analysis for binary additive stream ciphers focusses on analysis of the keystream generator and properties of the sequences it produces.

Keystream generators are referred to as bit-based or word-based, depending on the size of the output at each time interval. A bit-based keystream generator produces output sequences one bit at a time, whereas a word-based keystream generator produces output sequences one *n*-bit word at a time. Common word sizes are $n = 8$ or $n = 32$. For efficient implementation, word sizes may be selected to match particular computer architecture. The function used to combine the keystream with plaintext or ciphertext can still be addition modulo 2, regardless of the selected word size.

Cryptanalysis of a keystream generator for a stream cipher involves examining the structure of the generator and considering relationships between the internal state values and the output. For a good keystream generator, given knowledge of the structure of the generator and a segment of the keystream, a cryptanalyst should not be able to predict the rest of the keystream any better than merely guessing. That is, the keystream should appear to be a random sequence. Necessary, but not sufficient, requirements for cryptographically useful pseudo-random binary sequences are a long period, large linear complexity and good statistical properties. High linear complexity avoids an attack using the Berlekamp-Massey algorithm (Massey, 1969), which requires a length of keystream only twice the linear complexity of the sequence to produce the entire keystream. A bias in the distribution of zeroes and ones in the keystream can be used to reduce the unpredictability of the keystream sequence.In addition to these properties, it is necessary for the keystream generator to provide resistance to known styles of attack, including time-memory–data tradeoff attacks (Babbage, 1995; Golić, 1997), divide and conquer attacks, correlation attacks (Siegenthaler 1985) and algebraic attacks (Courtois and Meyer, 2003).

In some communication systems, it is possible that transmission errors occur which require that the entire message be re-sent. When a synchronous stream cipher is used, then a security requirement is that a different keystream sequence be used. Another common situation associated with the use of synchronous stream ciphers is that a message will be sent in frames or packets, each encrypted under the same key, k. Once again it is important that a different keystream sequence be used for each frame. For either of these situations, the same key can be used with the keystream generator to produce a different output sequence, provided that the initial states at the start of keystream generation is different. To achieve this, a re-keying algorithm is used to combine the secret key, k, with a publicly known initialisation vector. Let v_j denote the initialisation vector used for the i^{th} re-keying. The initialisation scheme for a stream cipher describes how the initial state of the keystream generator can be derived from k and v_j. The use of re-keying introduces a new attack scenario, as it is now possible for a cryptanalyst to access multiple related keystreams, produced under the same k but different known v_j, typically sequential or varying in only a few bits. The crypanalyst's task is then to recover k given a set of (v_j, z) pairs.

The authors of this paper have been involved in the process of designing several keystream generators for synchronous stream ciphers, including both bit-based and word based designs. These designs have been submitted to international projects such as the New European Scheme for Signatures, Integrity and Encryption (NESSIE, 1999) and the current eSTREAM project. In this paper, we review these designs and the design principles that were applied to provide good keystream properties and resistance to known attacks for each of these keystream generators. The LILI family of bit-based keystream generators is discussed first, followed by the word-based DRAGON cipher.

LILI KEYSTREAM GENERATORS

The LILI family of keystream generators for stream ciphers were introduced in (Simpson et. al, 2000). LILI keystream generators are simple and fast keystream generators that use two binary LFSRs and two functions to generate a pseudorandom binary keystream sequence. The components of the keystream generator can be grouped into two subsystems based on the functions they perform: clock control and data generation. Each subsystem consists of an LFSR and a function. The LFSR for the clock-control

subsystem is regularly clocked. The output of the clock-control subsystem is an integer sequence which controls the clocking of the LFSR within the data-generation subsystem. If regularly clocked, the data-generation subsystem is a simple nonlinearly filtered LFSR (Rueppel, 1986). Hence a LILI keystream generator may be viewed as a clock-controlled nonlinear filter generator.

The clock-control subsystem of the keystream generator uses a pseudorandom binary sequence produced by a regularly clocked LFSR, $LFSR_C$, of length L_C and a function, f_c, operating on the contents of k stages of $LFSR_C$ to produce a pseudorandom integer sequence, $c = \{c(t)\}_{t=1}$. For practical applications, it is assumed that the feedback polynomial of $LFSR_C$ is primitive and that the initial state of $LFSR_C$ is not the all zero state. Then $LFSR_C$ produces a maximum-length sequence of period $P_c = 2^{Lc} - 1$. At time instant t, the contents of a fixed set of k stages of $LFSR_C$ are input to f_c and the output of f_c is an integer $c(t)$, such that $c(t)$ is an element of $\{1,2,...2^k\}$. The function f_C is a bijective mapping from $\{0,1\}^k \rightarrow \{1,...,2^k\}$ so that the distribution of integers $c(t)$ is close to uniform. Thus c is a periodic integer sequence with period equal to P_c. For example, $f_c(x_1,...x_k) = 1 + x_1 + 2x_2 + ... + 2^{k-x}\}x_k$ is appropriate.

The variable parameters in the clock-control subsystem are L_c, the feedback function of $LFSR_C$, k, the positions of k stages of $LFSR_C$ used as inputs to the clocking function f_c and f_c itself.

The data-generation subsystem of the keystream generator uses the integer sequence c produced by the clock-control subsystem to control the clocking of a binary $LFSR, LFSR_D$, of length L_D. At time instant t, $LSFR_D$ is clocked $c(t)$ times. The contents of a fixed set of n stages of $LSFR_D$ are input to a nonlinear Boolean function, f_D. The binary output of f_D forms the keystream bit $z(t)$. After $z(t)$ is produced, $LSFR_D$ is clocked and the process repeated to form the keystream z.

If $LSFR_D$ is regularly clocked, then the data-generation subsystem is simply a nonlinear filter generator. It is assumed that the feedback polynomial of $LSFR_D$ is primitive and that the initial state of $LSFR_D$ is not the all zero state. Then $LSFR_D$ produces a maximum-length sequence of

period $P_D = 2^{Ld} - 1$. The output of a regularly clocked nonlinear filter generator is a periodic binary sequence, *g*, with period dividing P_D. In (Simpson, 1999) it is proven that, if $LSFR_D$ has a primitive feedback polynomial and nonzero initial state and either f_D is balanced or P_D is prime and f_D is not a constant function (zero or one), then the period of *g is* P_D.

The variable parameters in the data-generation subsystem are L_D, the feedback function of $LSFR_D$, n, the positions of n stages of $LSFR_D$ used as inputs to the filter function f_D and f_D itself. The function f_D should be balanced, highly nonlinear and offer some order of correlation immunity relative to the positions of the n stages used as inputs to f_D. The nonlinearity of a Boolean function is defined to be the minimum Hamming distance between the function and any affine function.

LILI Keystream Properties

The basic requirements; period, linear complexity and statistical properties; are addressed with respect to the LILI family of keystream generators. Let both $LSFR_C$ and $LSFR_D$ have primitive feedback polynomials and nonzero initial states. If either $2^{Ld} - 1$ is a prime and f_D is not a constant function, or if f_D is balanced and $2^{Lc-1}\left(2^k + 1\right) - 1$ is relatively prime to $2^{Ld} - 1$ (provided that $f_c\left(0,...0\right) = 1$, then the period of the output sequence *z* is given by the product $P_z = \left(2^{Lc} - 1\right)\left(2^{Ld} - 1\right)$. Note that this period implies that each distinct initial state results in the production of a distinct keystream. This avoids the reduction in keyspace which commonly occurs in keystream generators using irregular clocking, where several initial states produce the same keystream.

For LILI keystream generators, the output of a nonlinear filter generator with period $P_D = 2^{Ld} - 1$ or a divisor of P_D is nonuniformly decimated by means of a sequence with period $P_c = 2^{Lc} - 1$. For a nonuniformly decimated nonlinearly filtered $LSFR$ sequence, the maximal attainable linear complexity is the product of L'_d and P_c, where L'_d is the linear complexity of the (regularly clocked) nonlinearly filtered $LSFR_D$ sequence. The value of L'_d depends on the filter function and on the positions of stages used for its inputs but is very likely to be lower bounded by ${}^{Ld}C_r$, the number of

combinations formed from L_d things, taken *r* at a time, where *r* is the nonlinear algebraic order of the filter function. This conjectured linear complexity for a binary sequence produced by a LILI keystream generator is supported by experimental results in (Simpson, 2000).

The distribution of zeroes and ones in the output sequence of the keystream produced by a LILI keystream generator is also examined. Under regular clocking, one period of the sequence *d* produced by $LSFR_D$ when regularly clocked contains $2^{Ld-1}-1$ zeroes and 2^{Ld-1} ones. For a balanced filter function such that $f_D(0,...,0)=0$, a segment of length $2^{Ld}-1$ of the regularly clocked nonlinear filter generator output sequence *g* has the same distribution of zeroes and ones as *d*. When the clocking of $LSFR_D$ is under the control of $LSFR_C$ and when the period of z is $P_z=(2^{Lc}-1)(2^{Ld}-1)$, then each pair of $LSFR_C$ and $LSFR_D$ states occurs exactly once in a period of z. Therefore one period of z contains $(2^{Lc}-1)(2^{Ld-1}-1)$ zeroes and $(2^{Lc}-1)(2^{Ld-1})$ ones, thus maintaining the same proportion of zeroes and ones as in *d*.

LILI-128 KEYSTREAM GENERATOR

LILI-128 (Dawson, 2000) is a specific cipher from the LILI family of keystream generators, which was submitted as a candidate to the NESSIE process in 2000. The objective of the NESSIE process was to obtain a portfolio of strong cryptographic primitives for consideration as New European Security Standards. The LILI-128 keystream generator has an internal state size of 128 bits, where the lengths of $LSFR_C$ and $LSFR_D$ are 39 and 89 bits, respectively.

LILI-128 Description

The clock-control subsystem of the LILI-128 keystream generator is based on the 39-bit register $LSFR_C$. The feedback polynomial of $LSFR_C$ is the primitive polynomial $x^{39}+x^{35}+x^{33}+x^{31}+x^{17}+x^{15}+x^{14}+x^2+1$, and the initial state of $LSFR_C$ is not permitted to be the all zero state. It follows that $LSFR_C$ produces a maximum-length sequence of period $P_C=2^{39}-1$. The function f_C in the clock-control subsystem operates on the contents of

$k = 2$ stages of $LSFR_C$., stages 12 and 20. The function is given by $f_c(x_{12,}x_{20}) = 2(x_{12}) + x_{20} + 1$. Hence the output of the clock-control subsystem is an integer sequence of period $P_C = 2^{39} - 1$.

The data-generation subsystem of LILI-128 uses the integer sequence *c* produced by the clock-control subsystem to control the clocking of a binary LFSR, $LSFR_D$,of length register $L_D = 89$.

The feedback polynomial of $LSFR_D$ is the primitive polynomial $x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1$, and the initial state of $LSFR_D$ is not permitted to be the all zero state. It follows that, if regularly clocked $LSFR_D$ produces a maximum-length sequence of period $P_D = 2^{89} - 1$, which is a Mersenne Prime. The contents of a fixed set of $n = 10$ stages of $LSFR_D$ are input to a specially chosen Boolean function, f_D. The 10 inputs to f_D are taken from register positions according to this full positive difference set: $(0,1,3,7,12,20,30,44,65,80)$. The function f_D has been chosen to be balanced, highly nonlinear and to satisfy the third order of correlation immunity relative to the positions of the 10 stages used as inputs to f_D. The binary output of f_D is the keystream bit $z(t)$. After $z(t)$ is produced, the *LSFRs* are clocked and the process repeated to form the keystream *z*.

LILI-128 Re-Keying Algorithm

The initial NESSIE call for submissions did not require a process for re-keying, so the initialisation process in the original submission is merely key loading: the 128 bit key is used directly to form the initial values of the two shift registers, from left to right: the first 39 bits in $LSFR_C$ then the remaining 89 bits in $LSFR_D$. During the NESSIE process, a re-keying scheme which accepted the 128-bit secret key *k* and an initialisation vector v_i of up to 128 bits was requested. For efficiency, it was decided to use the structure of the LILI-128 keystream generator in the re-keying process.

The LILI-128 re-keying process is as follows. Firstly, the initial state for the keystream generator is formed by binary addition modulo two (XOR) of the 128-bit key *k* and the 128 bit initialisation vector v_i (for a shorter initialisation vector, concatenation of v_i to form a 128-bit string is required). Secondly, the keystream generator is run to produce an output segment, of which the leading *b* bits are discarded. Thirdly, the next 128 bits are loaded

into the *LSFRs*. The second and third steps are repeated a times, to form the initial state for actual keystream generation. Suggested values for a and b are either $a=1$ and $b=128$, or $a=2$ and $b=0$. The total computation time required for re-keying is given by $t=a(b+128)$, so for either of the suggested options, re-keying takes approximately the amount of time required to perform keystream generation of 256 bits.

In the rare event that either register is initialised to an all zero state, then that initial state is declared invalid. All distinct valid initial states produce different keystreams, and there are no known weak keys.

LILI-128 Keystream Properties

Keystream sequences produced by the LILI-128 keystream generator have a period of $(2^{39}-1)(2^{89}-1)\approx 2^{128}$. Note that this period implies that each distinct initial state results in the production of a distinct keystream. Secondly, the linear complexity of these keystream sequences is conjectured to be at least 2^{68}, so that at least 2^{69} consecutive bits of known plaintext are required for the Berlekamp-Massey attack to be successful. This is an infeasible amount of text to collect. Finally, the distribution of ones and zeros in the keystream is the same as that in the underlying LFSR sequence.

Attacks on LILI-128

There are three major published attacks on the LILI-128 keystream generator. In chronological order, these are a time-memory trade-off attack (Saarinen, 2002), algebraic attacks (Courtois and Meier, 2003) and a correlation attack (Molland, 2004). The following paragraphs provide a brief outline of each of these attacks.

A general time-memory-data tradeoff attack by Biryukov and Shamir (2000) can be applied to the LILI-128 keystream generator. One possible tradeoff requires precomputation time and computation time equal to about 2^{86} table lookups, and memory in 128-bit words equal to a known keystream length of about 2^{43} bits. The time-memory trade-off attack proposed by Saarinen (Saarinen, 2002) reduces the complexity of the attack by ignoring $LSFR_C$ in the precomputation, and forming a table of pairs of 89-bit keys and 45-bit keystream segments formed by taking the output bits of f_D from a regularly clocked $LFSR_D$ at intervals equivalent to the number of times $LSFR_D$ is clocked in one period of $LSFR_C$. During the runtime phase, the keystream

bits are sampled at intervals equivalent to the period of $LSFR_C$, and the stored table is searched for a match with the resultant string. This approach reduces the computational complexity of both the precomputation and real-time phases of the attack to about 2^{56}, and requires storage in memory for a table of 2^{89} 45-bit words. The reduction is achieved by an increase in the amount of known keystream required to a length of about 2^{46} bits.

Algebraic attacks on stream ciphers involve solving a system of multivariate polynomial equations relating the initial state of the stream cipher to the keystream produced, to recover the values of the initial internal state bits. These attacks have been highly effective against keystream generators based on regularly clocked bit-based *LSFRs*. Courtois and Meier (2003) suggest applying such an attack to the LILI-128 keystream generator in a divide and conquer style, by firstly guessing the 39-bit state of $LSFR_C$ generating the clock control sequence, and then applying the algebraic attack to the appropriately positioned bits of the keystream sequence to recover the initial state of $LSFR_D$. An inconsistent set of equations indicates an error in guessing the initial state for $LSFR_C$. Such an attack requires a known keystream length of about 2^{18} bits, storage in memory of about 2^{43} bits and a runtime complexity of about 2^{96} memory lookups. An alternative algebraic attack uses a similar approach to Saarinen in ignoring the clock control provided by $LSFR_C$ and running a fast algebraic attack. Requirements for this attack are a known keystream length of about 2^{60} bits, storage in memory of about 2^{24} bits and a runtime complexity in the order of about 2^{31} memory lookups.

The correlation attack on LILI-128 by Molland (Molland, 2004) uses the correlation properties of the Boolean function f_D. A slight bias related to the output of the function permits a distinguishing attack, so that the output sequence produced by a regularly clocked nonlinear filter generator using this filter function can be distinguished from a truly random binary sequence, given enough keystream bits. The attack on LILI-128 consists of guessing the initial state of the clock-control register $LSFR_C$ producing the clock control sequence and positioning the known keystream bits as they would be in a regularly clocked nonlinear filter generator. Then the identified distinguisher is used to determine if the guessed initial state of $LSFR_C$ was correct. Once the initial state of $LSFR_C$ is determined, the initial state of $LSFR_D$ is obtained by running a fast correlation attack. According to Molland, the requirements for this attack are a known keystream length of

about 2^{29} bits, storage in memory of about 2^{28} bits and a runtime complexity in the order of about 2^{57} operations.

We note that although these attacks are theoretically possible; that is, they require less effort than exhaustive key search, for the most part they are not practical now or in the near future. For any keystream generators for stream ciphers the only way to prevent possible attacks which are faster than exhaustive key search is to use an initial state which is greater than the key size along with a secure initialisation scheme which will transform the secret key k and some other public information called an initialisation vector, v_i, into a longer initial state.

LILI-II KEYSTREAM GENERATORS

Hypothesised attacks on LILI-128, and the request for a re-keying proposal prompted a review of the LILI-128 parameters to ensure provable security properties could be maintained while achieving an effective key size of 128 bits. LILI-II (Clark et al, 2002) is a specific cipher from the LILI family of keystream generators which achieves this security goal. The cipher has an internal state size of 255 bits, where the lengths of $LSFR_C$ and $LSFR_D$ are 128 and 127 bits, respectively.

LILI-II Description

The clock-control subsystem of the LILI-II keystream generator is based on a 128-bit register $LSFR_C$. The feedback polynomial of $LSFR_C$ is a particular primitive polynomial of degree 128 which is not sparse, and the initial state of $LSFR_C$ is not permitted to be the all zero state. It follows that $LSFR_C$ produces a maximum-length sequence of period $P_c = 2^{128} - 1$. The function f_C in the clock-control subsystem operates on the contents of $k = 2$ stages of $LSFR_C$, stages 0 and 126. The function is given by $f_c(x_{0,} x_{126}) = 2(x_0) + x_{126} + 1$. Hence the output of the clock-control subsystem is an integer sequence of period $P_c = 2^{128} - 1$.

The data-generation subsystem of LILI-II uses the integer sequence c produced by the clock-control subsystem to control the clocking of a binary $LSFR$, $LSFR_D$,of length register $L_D = 127$. The feedback polynomial of $LSFR_D$ is a particular primitive polynomial of degree 127 which is not

sparse, and the initial state of $LSFR_D$ is not permitted to be the all zero state. It follows that, if regularly clocked $LSFR_D$ produces a maximum-length sequence of period $P_D = 2^{127} - 1$, which is a Mersenne Prime. The contents of a fixed set of $n = 12$ stages of $LSFR_D$ are input to a specially chosen Boolean function, f_D. The 12 inputs to f_D are taken from register positions which form a full positive difference set: $(0,1,3,7,12,20,30,44,65,80,96,122)$. The function f_D has been chosen to be balanced, highly nonlinear and has first order correlation immunity. The function f_D has nonlinearity of 1992, and an algebraic order of 10. The binary output of f_D is the keystream bit $z(t)$. After $z(t)$ is produced, the $LSFRs$ are clocked and the process repeated to form the keystream z. A full description of the LILI-II keystream generator, including the primitive polynomials associated with the two $LSFRs$ is given in (Clark, *et al.* 2002).

LILI-II Re-Keying Algorithm

A re-keying scheme which accepted the 128-bit secret key k and an initialisation vector v_j of up to 128 bits was requested. If the initialisation vector has length less than 128 bits, then multiple copies of the vector are to be concatenated and truncated as required, to form a 128-bit vector. As for LILI-128, it was decided for efficiency to use the structure of the LILI-II keystream generator in the re-keying process.

The LILI-II re-keying process uses the generator itself twice. Firstly, the initial state of $LSFR_C$ is obtained by XORing the two 128-bit binary strings k and v_i, and the initial state of $LSFR_D$ is obtained by deleting the first bit of k and the last bit of v_i and XORing the two resulting 127-bit binary strings. Secondly, the cipher is run to produce an output string of length 255 bits. For the second application of the cipher, the first 128 bits of this output string are used to form the initial state of $LSFR_C$ and the remaining 127 bits are used to form the initial state of $LSFR_D$. The cipher is run again to produce an output string of length 255 bits. The output from this second application is used to form the initial state of the keystream generator at the beginning of keystream production. As previously, the first 128 bits form the initial state of $LSFR_C$, and the remaining 127 bits form the initial state of $LSFR_D$.

In the rare event that either register is initialised to an all zero state, then that initial state is declared invalid. All distinct valid initial states produce different keystreams, and there are no known weak keys.

LILI-II Keystream Properties

Firstly, keystream sequences produced by the LILI-II keystream generator have a period of $\left(2^{128}-1\right)\left(2^{127}-1\right) \approx 2^{255}$. Note that this period implies that each distinct initial state results in the production of a distinct keystream. Also, this greatly exceeds the length of any practical plaintext, rendering any attacks based on a short period infeasible. Secondly, the linear complexity of these keystream sequences is conjectured to be at least 2^{175}, so that at least 2^{176} consecutive bits of known plaintext are required for the Berlekamp-Massey attack to be successful. This is an infeasible amount of text to collect. Finally, the distribution of ones and zeros in the keystream is the same as that in the underlying *LSFR* sequence.

LILI-II Attack Resistance

The large internal state size of 127 + 128 = 255 bits causes any of the tradeoffs for the general time-memory-data tradeoff attacks to be worse than exhaustive key search. For example, for the time-memory attacks described in (Babbage, 1995), where the product of the time and memory requirements is equivalent to the state space, and the amount of keystream required is equivalent to the time, a $2^{128}.2^{127} = 2^{255}$ tradeoff could be used, although this requires time equivalent to exhaustive key search, and an excessive amount of memory and known keystream.
In any case, the use of the initialisation scheme (the key-loading/re-keying algorithm) to expand the 128-bit secret key into a 255 bit initial state renders the time-memory attacks on LILI-II infeasible, as their performance is at best, no better than exhaustive key search. By employing the LILI-II algorithm itself during re-keying, we take advantage of both the known security properties of the algorithm and also its fast implementation. We have considered also the possibility of related key attacks using multiple key-streams, but can identify no attack in the re-keying scenario which performs better than exhaustive key search.

The large internal state size also provides resistance against the standard divide and conquer attacks, which target one or other of the *LSFRs*. We conjecture that the complexity of any divide and conquer attacks on LILI-II

exceeds 2^{128} operations, and also requires a substantial amount of known keystream bits.

Taken together, these results indicate that LILI-II is a secure synchronous stream cipher, in that there are no currently known attacks on the cipher which are more feasible than exhaustive key search.

LILI Keystream Generator Implementation Considerations

There are several internal differences between the implementation of LILI-128 and that of LILI-II. These differences include using Galois structure (rather than traditional Fibonacci style), for the *LSFR* state transitions, the increase in the size of the registers and increasing the number of inputs to the filter function from 10 to 12. These aspects have different effects on the speed of the design in software. LILI-II is less efficient in software than LILI-128 mainly due to the larger *LSFRs* and larger Boolean function which are used in the design to increase security. However, in hardware LILI-II offers the same high speed as LILI-128.

DRAGON KEYSTREAM GENERATOR

Dragon (Chen, et al. 2005) is a word-based stream cipher designed with both security and efficiency in mind. The cipher was submitted to the ECRYPT eSTREAM project in April, 2005 and is currently one of the Phase 2 Focus group ciphers. Dragon is constructed using a single word based non-linear feedback shift register (NLFSR) and a nonlinear filter function with memory. Dragon uses a variable length key (128 or 256 bits) and initialisation vector of 128 or 256 bits, and produces 64 bits of keystream per iteration. At the heart of Dragon are two highly optimised 8×32 s-boxes. Dragon uses simple operations on 32-bit words to provide a high degree of efficiency in a wide variety of environments, making it highly competitive when compared with other word based stream ciphers.

DRAGON description

Dragon is constructed using a single word-based NLFSR. The register consists of thirty-two stages, each of which holds a 32-bit word. Thus the length of the register is 1024 bits. There is also a 64-bit memory component, *M*, so the total internal state size is 1088 bits. The state update function, *F*, is used in both key setup and keystream generation operations. The cipher can be used with either 128-bit key and initialisation vectors (DRAGON-128), or with 256-bit key and initialisation vectors (DRAGON 256).

The *F* function is a reversible mapping from 192 bits (six 32-bit words) to 192 bits. That is, it takes six 32-bit words as input, and produces six 32-bit words as output. Let *a*, *b*, *c*, *d*, *e* and *f* denote the six 32-bit words of input to *F* and *a'*, *b'*, *c'*, *d'*, *e'* and *f'* denote the six 32-bit words output. The *F* function has six component functions: $G_1, G_2, G_3, H_1, H_2,$ and H_3. The *G* and *H* subfunctions, constructed from two 8x32-bit S-boxes, are used to provide algebraic completeness and high nonlinearity. Details of the subfunctions and the S-boxes are given in (Chen, et al. 2005). A network of modular additions (addition modulo 2^{32}, denoted +) and binary additions (addition modulo 2, denoted $\oplus$) are used for diffusion in the *F* function. The *F* function can be divided into three parts: pre-mixing, substitution and post-mixing. Each part is designed for parallelisation, permitting speedy operation. These layers are detailed below.

In the premixing layer, two sets of operations can be performed in parallel:

Set 1:	$B = b \oplus a$	$d = d \oplus c$	$f = f \oplus e$
Set 2:	$C = c + b$	$e = e + d$	$a = a + f$

This is followed by the substitution layer (applying the s-boxes):

Set 3:	$D = d \oplus G_1(a)$	$f = f \oplus G_2(c)$	$b = b \oplus G_3(e)$
Set 4:	$A = a \oplus H_1(b)$	$c = c \oplus H_2(d)$	$e = e \oplus H_3(f)$

A final layer of mixing operations is performed:

Set 5:	$d' = d + a$	$f' = f + c$	$b' = b + e$
Set 6:	$c' = c \oplus b$	$e' = e \oplus d$	$a' = a \oplus f$

Dragon has a simple key and re-keying strategy, using the key k and a known initialisation vector v_i. We briefly outline the rekeying scheme for Dragon -128 here. More extensive details, and the re-keying scheme for Dragon-256 may be obtained from (Chen, *et al.* 2005). The 1024-bit NLFSR is divided into eight 128-bit words, labelled W_0 to W_7. This is initially filled by concatenating the key, the initialisation vector, and simple functions of these such as $k = k \oplus v_i$. The memory component is initialised to a constant value. Following this, 16 iterations of the *F* function are performed, updating both the NLFSR and *M* in each round. The use of existing cipher

components in the initialisation scheme simplifies analysis and increases implementation efficiency.

For keystream generation, inputs are taken from six of the thirty-two 32-bit stages of the NLFSR. The selected stages are 0, 9, 16, 19, 30, and 31. These inputs form a full positive difference set. The 64-bit memory component *M* acts as a counter during keystream generation, with the initial value of *M* for keystream generation being the value of *M* at the end of the initialisation process. Inputs from the six 32-bit NLFSR stages and from *M* are used to form the six 32-bit inputs to the *F* function. Four of the six 32-bit outputs are used to update the NLFSR and the remaining two are used to produce the 64-bit output. This is repeated to produce the desired length of keystream.

DRAGON Design Principles

The key setup and keystream generation of Dragon both use the *F* function, for ease of implementation and for increased efficiency. However, the operations performed in key setup are deliberately different to those in keystream generation, so that the mapping of internal state to feedback is different. Another design consideration during initialisation was to ensure that no 256-bit key-initialisation vector pair will result in an initial state for Dragon that could be produced by an arbitrary 128-bit pair. This precludes a crypanalyst reducing the search space for a 256 bit key to that of a 128-bit key.

DRAGON Keystream Properties

Given the use of a 1024-bit NLFSR, the expected period for the internal state is around 2^{512}, assuming a pseudo-random mapping. With the additional memory component the expected period is increased to 2^{576}. For cryptographic use, a clear lower bound on the period is more important than an average value. The inclusion of the 64-bit counter, *M*, provides a guaranteed lower bound on the period of 2^{64}. Re-keying after the production of 2^{64} bits of keystream is recommended, and avoids the possibility of keystream collision attacks.

Statistical tests provided by the CRYPT-X package were performed on the keystream produced by the Dragon cipher. The frequency, binary derivative, change point, subblock and runs tests were performed on 30 streams of Dragon output, each of length eight megabytes. The sequence and linear complexity tests were executed for the thirty streams, each of length two hundred kilobytes. The Dragon keystream generator passed all of these tests.

Thus keystream produced by the Dragon generator appears to be suitably pseudo-random.

Weak keys are those that bypass some operations of the cipher, that is, those keys for which the operations have negligible effect in calculating the feedback or the output. Dragon is designed to avoid weak keys. Use of the large word-based NLFSR and of functions with no fixed points has avoided the need to place restrictions on the internal state values (for LFSRs, it is common to forbid the all zero state, and this needs to be incorporated into re-keying schemes). Also the 16 iterations of the state update function during ke-keying ensures that specific differentials cannot be traced through all rounds. This provides strong evidence that there are no weak keys for Dragon.

Possible Attacks on the DRAGON Keystream Generator

The use of initialisation vectors to produce multiple keystreams from a single master key gives the cryptanalyst the possibility of using multiple keystreams and the corresponding known initialisation vector to deduce information about the underlying master key. This may be possible if the initial state at the start of keystream generation can be readily expressed in terms of the unknown key bits and the known initialisation vector bits, so that the effect of small known changes in the initial state can be readily observed. However, the re-keying scheme for Dragon mixes each key bit into all words of the initial state by applying the highly nonlinear F function in each of 16 iterations. During initialisation, the F function output is used to update both the NLFSR and the memory, so the value of the memory is difficult to determine after the first iteration. The contents of the memory unit are also incorporated into the inputs to the F function during initialisation, increasing the difficulty for an attacker.

Time-Memory tradeoff attacks rely on precomputation to reduce the effort required for a key recovery attack. In a precomputation phase, the attacker calculates a table of keys or internal states and the associated keystream prefix. In the real time attack phase, the attacker attempts to match observed keystream to table entries, and thus recover the internal state and/or key bits. The large internal state space (1088 bits) of the Dragon keystream generator in comparison to the key size, coupled with the initialisation scheme means that an attacker would have to precompute a table for each initialisation vector. Any tradeoff of time, memory or keystream requirement will be much worse than exhaustive key search.

The stages of the NLFSR used to provide input to the state update function form a full positive difference set. This is useful in providing resistance to guess and determine style attacks, as for any guessed subset of inputs at a fixed point in time, at most one of these values will also be an input at the next time interval. Thus an attacker must guess increasingly more input values. The rapid increase in the number of possible guess pathways makes this approach infeasible. This attack strategy cannot be more effective than exhaustive key search.

A distinguishing attack is an attack in which the output sequence from a stream cipher can be statistically distinguished from a truly random sequence. That is, there is some property of the keystream sequence that an attacker can identify which indicates that the sequence has been produced by a particular pseudorandom keystream generator. Englund and Maximov (2005) describe a distinguishing attack against Dragon-256 which requires 2^{155} keystream words produced from a single key-initialisation vector pair. This keystream requirement greatly exceeds the permitted maximum keystream length of 2^{64} bits, and so is of theoretical interest only, and does not constitute a security compromise. There are two variants of the attack, one with complexity of 2^{187} operations and memory requirements of 2^{32} words, and a second with complexity of 2^{155} operations and memory requirements of 2^{96} words. Disregarding the excessive keystream requirement, neither attack is better than exhaustive keysearch for Dragon-128. In any case, the attack remains a distinguishing attack, and cannot be extended to key recovery.

Dragon Keystream Generator Implementation Considerations

Dragon is designed to be efficient in both software and hardware, in terms of both throughput and implementation footprint. The 32-bit word size was selected to match that of the ubiquitous Intel Pentium family, as it leads to the best software efficiency on this platform. The design allows a high degree of parallelisation in hardware, as the operations can be divided into three groups, each operating on two inputs.

IMPLEMENTATION

The bit-based LILI keystream generators and the word-based Dragon design have very different structures, with an increase in complexity accompanying the shift to word-based design. Both designs can be implemented in either hardware or software. The bit-based design of the

LILI generators provide a better performance in hardware, whereas the word-based Dragon generator is much faster in software, as expected. For comparison, software performance figures on a 3.2 GHz Intel Pentium 4 are given in the following table for the keystream generators LILI-128, LILI-II and Dragon, and for the block cipher AES. AES is the current block cipher encryption standard. As there is no current stream cipher standard, and a block cipher can be used (in an appropriate mode) for keystream generation, a useful stream cipher should be able to provide greater throughput than AES to be considered useful. On this basis, we recommend the use of Dragon rather than LILI for software implementation.

Keystream generator	Throughput
LILI-128	72 Mbps
LILI-II	66 Mbps
Dragon	3820 Mbps
AES	1828 Mbps

CONCLUSION

In this paper we described and discussed the LILI family of keystream generators (including the particular instances LILI-128 and LILI-II) and also the Dragon keystream generator, submissions to NESSIE and eSTREAM respectively. The principles on which design choices were based are outlined.

The progression of the designs over time shows the changing landscape for stream cipher design. Most keystream generator designs for stream ciphers proposed prior to the year 2000 were bit-based designs, like the LILI family. Commonly, they used the key directly as the initial state, and re-keying to make use of a secret master key with multiple known initialisation vectors was uncommon. The application of stream ciphers to new environments such as mobile telephony and internet applications has changed this. The call for initialisation and re-keying schemes to be added to the NESSIE submissions reflects the importance of these aspects in design specifications

The increasing application of encryption in software rather than hardware has been a driving force in the move toward word-based designs. Many of the designs submitted to the recent eSTREAM project are word-based stream ciphers, like Dragon. The only bit-based designs submitted to eSTREAM are submissions under the hardware profile.

As the complexity of proposed stream cipher designs increases, it becomes difficult to provide theoretical support for security claims, even regarding basic security properties. Both LILI and Dragon designs have been subject to public scrutiny. Both LILI-II and Dragon ciphers are efficient and secure keystream generators for stream ciphers. We recommend the use of LILI-II for hardware applications and Dragon for software implementation. Further investigation into the security provided by these designs is welcomed.

REFERENCES

Babbage, S. 1995. A space/time trade off in exhaustive search attacks on stream ciphers. *European Convention on Security and Detection, {IEE} Conference Publication No. 408*, May 1995.

Biryukov, A. and Shamir, A. 2000. Cryptanalytic Time/Memory/Data tradeoffs for stream ciphers. *Asiacrypt 2000*, Vol. 1976 of *Lecture Notes in Computer Science*, pp. 1 - 13, Springer-Verlag.

Clark, A., Dawson, E., Fuller, J., Golic, J., Lee, H., Millan, W., Moon, S. and Simpson, L. 2002. The LILI-II Keystream Generator. *Australasian Conference on Information Security and Privacy -ACISP 2002*, vol 2384 of *Lecture Notes in Computer Science*, pp. 25 - 39, Springer-Verlag.

Chen, K., Henricksen, M., Millan, W., Fuller, J., Simpson, L., Dawson, E., Lee, H. and Moon, S. 2005. Dragon: A Fast Word Based Stream Cipher. *Information Security and Cryptology – ICISC2004.* vol 3506 of *Lecture Notes in Computer Science*, pp.35 - 50, Springer-Verlag. Also *ECRYPT eSTREAM submission*, available at: http://www.ecrypt.eu.org/stream/dragonp2.html

Courtois, N. and Meier, W. 2003. Algebraic attacks on stream ciphers with linear feedback. In *Advances in Cryptology – EUROCRYPT 2003*, vol 2656 of *Lecture Notes in Computer Science*, pp. 176 - 194, Springer-Verlag.

Dawson, E., Clark, A., Golić, J, Millan, W., Penna, L. and Simpson, L. 2000. The LILI-128 Keystream Generator. *NESSIE submission*, in the proceedings of the *First Open NESSIE Workshop (Leuven, 2000)* and available at: http://www.cryptonessie.org

Englund, H. and Maximov, A. 2005. Attack the Dragon. *ECRYPT eSTREAM submission*, available at: http://www.ecrypt.eu.org/stream/papersdir/ 062.pdf. submitted September 2005.

Golić, J. 1997. Cryptanalysis of Alleged A5 stream cipher. *Advances in Cryptology – EUROCRYPT'97*, vol 1233 of *Lecture Notes in Computer Science*, pp. 239-255. Springer-Verlag.

Massey, J. 1969. Shift-Register Synthesis and BCH Decoding. *IEEE Trans. Inform. Theory*, IT-15:122-127, January 1969.

Molland, H and Telleseth, T. 2004. An improved correlation attack against irregular clocked and filtered keystream generators. In *Advances in Cryptology – CRYPTO 2004*, vol 3152 of *Lecture Notes in Computer Science*, pp. 373-389, Springer-Verlag.

NESSIE. 1999. *New European Schemes for Signatures, Integrity and Encryption*. Project within the Information Society Technologies (IST) Programme of the European Commision. URL: https://www.cosic.esat.kuleuven.be/nessie/

Rueppel, R. 1986. *Analysis and design of stream ciphers*. Springer-Verlag, Berlin.

Saarinen, M. 2002. A Time-Memory Tradeoff Attack Against LILI-128.' *Fast Software Encryption – FSE 2002* vol 2365 of *Lecture Notes in Computer Science*, pp. 231-236, Springer-Verlag.

Siegenthaler, T. 1985. Decrypting a Class of Stream Ciphers Using Ciphertext Only.' *IEEE Trans. Computers*, **C-34**(1): 81--85.

Simpson, L., Dawson, E. Golić, J and Millan, W. 2000. LILI Keystream Generator'.*Proceedings of the Seventh Annual Workshop on Selected Areas in Cryptology – SAC'2000*, volume~2012 of *Lecture Notes in Computer Science*, pp. 248-261, Springer-Verlag.