Dragon: A Fast Word Based Stream Cipher*

K. Chen[1], M. Henricksen[1], W. Millan[1], J. Fuller[1], L. Simpson[1], E. Dawson[1], H. Lee[2], and S. Moon[3]

[1] Information Security Research Centre, Queensland University of Technology, GPO Box 2434, Brisbane Qld 4001, Australia
{k.chen, m.henricksen, b.millan, j.fuller, lr.simpson, e.dawson}@qut.edu.au

[2] School of Internet Engineering, Dongseo University, San 69-1, Churye-2 Dong, SaSang-Ku, Pusan 617-716, Korea
hjlee@dongseo.ac.kr

[3] School of Electronic and Electrical Engineering, Kyungpook National University, 1370, Sankyuk-dong, Taegu 702-701, Korea
sjmoon@knu.ac.kr

Abstract. This paper presents Dragon, a new stream cipher constructed using a single word based non-linear feedback shift register and a non-linear filter function with memory. Dragon uses a variable length key 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. The components of Dragon are designed to resist all known attacks.

1 Introduction

Traditionally stream cipher design has focussed on bit based linear feedback shift registers (LFSRs), as these are well studied and produce sequences which satisfy common statistical criteria. In these ciphers, non-linearity is introduced into the keystream either by some type of non-linear combining function or filter function, or by irregular clocking, or both. However, bit based LFSRs are notoriously slow in software. Each iteration of the cipher's update function produces only one bit of keystream. Sparse LFSR feedback functions may be exploited in an attack, but increasing the number of feedback taps results in a decrease in efficiency. Also, the security of some LFSR based stream ciphers is threatened by algebraic attacks [6].

Word based stream ciphers may provide a solution to the security-efficiency tradeoff. These produce many times the amount of keystream per iteration than do bit-based LFSRs, depending on the word size. The word size used for current

* This research was supported by Australian Research Grant No DP0450920 and South Korean University IT Research Center Project for Mobile Network Security Technology Research Center.

word based stream cipher proposals range from 8-bit words for RC4 to 32-bit words for Turing [16]. Many of these ciphers are very fast in software, outperforming even fast block ciphers like the Advanced Encryption Standard [15]. Although it is easy to assess the speed of these word based stream ciphers, it is difficult to quantify their security precisely.

This paper presents Dragon, a new word based stream cipher designed with both security and efficiency in mind. Dragon uses a word based non-linear feedback shift register (NLFSR), in conjunction with a non-linear filter to produce keystream as 64-bit words. Dragon has a throughput of gigabits per second in both modern software and hardware, and requires little more than four kilobytes of memory, so is suitable for use in constrained environments. Not only is Dragon fast for keystream generation, it is also very efficient when rekeying. This makes Dragon especially suitable for applications that require frequent rekeying, such as mobile and wireless communications.

Dragon can be considered an evolution of the output feedback mode (OFB) of block ciphers. The modifications overcome a shortcoming of block ciphers in OFB mode: the output keystream is also the feedback to the internal state. We have analysed the security of Dragon using modern cryptanalytic techniques, and believe it is suitable for use as a secure cryptographic primitive. Collision attacks based on the birthday paradox can exploit this knowledge of the feedback. These attacks are prevented in the Dragon cipher by producing separate output and feedback words from the update function. Also, ciphers with small internal states are easily attacked by time/memory/data tradeoff attacks [3]. A minimum requirement to overcome these type of attacks is to have an internal state size at least twice the designed security. Time/memory/data tradeoff attacks are prevented in the Dragon cipher by having a large internal state. To increase the difficulty of guess and determine attacks [10], Dragon selects taps from the NLFSR according to a Full Positive Difference Set (FPDS).

Section 2 presents the specification of the cipher. Section 3 describes the design decisions behind the Dragon algorithm. Section 4 and 5 includes a security analysis of Dragon using modern cryptanalytic techniques. Section 6 discusses the performance of Dragon in software and hardware, and associated implementation issues.

2 Specification of Dragon

Dragon is a stream cipher constructed using a single word based NLFSR. Dragon has a large NLFSR of 1024 bits, an update function, denoted F, and a 64-bit memory, denoted M. Dragon-256 uses a secret master key of 256 bits, and a publicly known initialisation vector (IV), also of 256 bits to accommodate rekeying scenarios, while Dragon-128 uses 128-bit key and IV. The F function, which is called once per round, manipulates the internal state to generate 64 bits of pseudo-random keystream. The cipher's key setup converts the master key and initialisation vector for use in Dragon's large internal state.

Input = { a, b, c, d, e, f }			
Pre-mixing Layer:			
1. $b = b \oplus a$;	$d = d \oplus c$;	$f = f \oplus e$;	
2. $c = c \boxplus b$;	$e = e \boxplus d$;	$a = a \boxplus f$;	
S-box Layer:			
3. $d = d \oplus G_1(a)$;	$f = f \oplus G_2(c)$;	$b = b \oplus G_3(e)$;	
4. $a = a \oplus H_1(b)$;	$c = c \oplus H_2(d)$;	$e = e \oplus H_3(f)$;	
Post-mixing Layer:			
5. $d' = d \boxplus a$;	$f' = f \boxplus c$;	$b' = b \boxplus e$;	
6. $c' = c \oplus b$;	$e' = e \oplus d$;	$a' = a \oplus f$;	
Output = { a', b', c', d', e', f' }			

Table 1. Dragon's F Function

2.1 Dragon's State Update Function (*F* Function)

The F function is used in both key setup and keystream generation. The F function is a reversible mapping of 192 bits (six 32-bit words) to 192 bits. It takes six 32-bit words as input and produces six 32-bit words as output. In Table 1 the input words are denoted a, b, c, d, e, f and the output words a', b', c', d', e', f'. The F function has six component functions denoted G_1, G_2, G_3, H_1, H_2 and H_3, as described below. The G and H functions provide algebraic completeness [11] and high non-linearity. A network of modular and binary additions are used for diffusion in the F function. It can be divided to three parts: pre-mixing, substitution, and post-mixing. Each step is designed to allow for parallelisation, giving Dragon its speed. The F function is shown in Table 1 where $\oplus$ denotes XOR and $\boxplus$ denotes addition modulo 2^{32}.

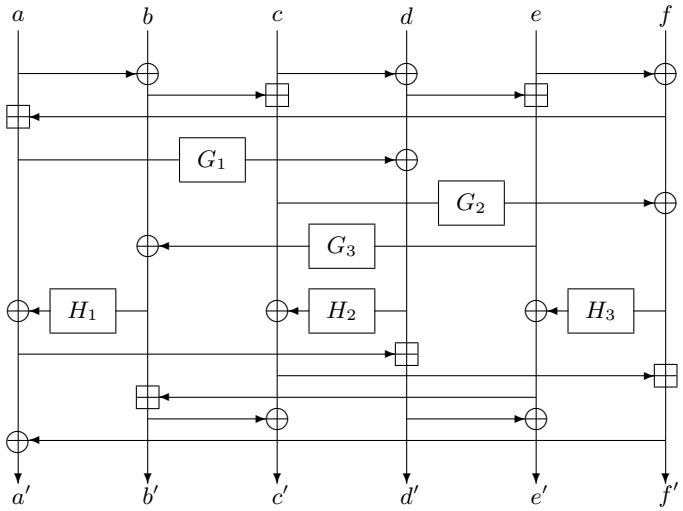

Fig. 1. F Function

***G* and *H* Functions** The G and H functions are constructed from two 8×32-bit s-boxes, S_1 and S_2 to form virtual 32×32 s-boxes. The G functions contains three S_1s and one S_2, while the H functions have three S_2s and one S_1. S_1 and S_2 are included in Appendix B. The 32-bit input is broken into four bytes ($x = x_0 \| x_1 \| x_2 \| x_3$. Each byte is passed through an 8×32 s-box and the four 32-bit outputs combined using binary addition.

G and H functions are defined as

$$G_1(x) = S_1(x_0) \oplus S_1(x_1) \oplus S_1(x_2) \oplus S_2(x_3)$$

$$G_2(x) = S_1(x_0) \oplus S_1(x_1) \oplus S_2(x_2) \oplus S_1(x_3)$$

$$G_3(x) = S_1(x_0) \oplus S_2(x_1) \oplus S_1(x_2) \oplus S_1(x_3)$$

$$H_1(x) = S_2(x_0) \oplus S_2(x_1) \oplus S_2(x_2) \oplus S_1(x_3)$$

$$H_2(x) = S_2(x_0) \oplus S_2(x_1) \oplus S_1(x_2) \oplus S_2(x_3)$$

$$H_3(x) = S_2(x_0) \oplus S_1(x_1) \oplus S_2(x_2) \oplus S_2(x_3)$$

Input = { K, IV } (256-bit) Input = { k, iv } (128-bit)	
1.	$W_0 \| ... \| W_7 = K \| K \oplus IV \| \overline{K \oplus IV} \| IV$ (256-bit) $W_0 \| ... \| W_7 = k \| k' \oplus iv' \| iv \| k \oplus iv' \| k' \| k \oplus iv \| iv' \| k' \oplus iv$ (128-bit)
2.	$M =$ 0x0000447261676F6E
Perform steps 3-8 16 times	
3.	$a \| b \| c \| d = (W_0 \oplus W_6 \oplus W_7)$
4.	$e \| f = M$
5.	$\{a', b', c', d', e', f'\} = F(a, b, c, d, e, f)$
6.	$W_0 = (a' \| b' \| c' \| d') \oplus W_4$
7.	$W_i = W_{i-1}$, for $i = 7$ down to 1 (shifting the state by one word)
8.	$M = e' \| f'$
Output = { $W_0 \| ... \| W_7$ }	

Table 2. Dragon's Key Initialisation function

2.2 Initialisation

Dragon can be used with two different key and initialisation vector lengths: 128-bit and 256-bit. We denote the 256-bit key and initialisation vector K and IV respectively. The 128-bit key and initialisation vector are denoted k and iv.

Dragon has a simple keying (and rekeying) strategy using the key and the publicly known initialisation vector. The 1024-bit internal state is divided into eight 128-bit words, labelled W_0 to W_7. The internal state is initially filled by concatenating the key and the initialisation vector. The state initialisation process

makes extensive use of the F function. The initialisation involves 16 iterations of the F function as shown in Table 2 where $\overline{x}$ denotes the complement of x and x' denotes the swapping of the upper half and the lower half of x. To protect against unknown future attacks, and against attacks that require large amounts of keystream, the cipher should be rekeyed at least once for every 2^{64} bits of keystream generated. The use of existing components for both initialisation and keystream generation simplifies analysis and increases implementation efficiency.

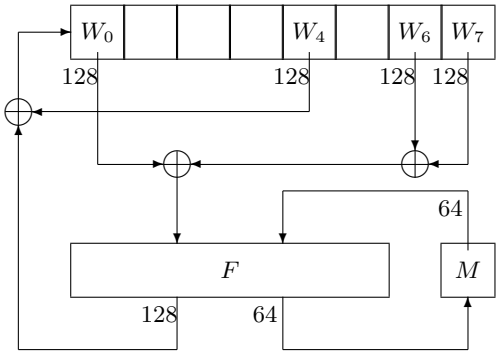

Fig. 2. Initialisation

Input = { $B_0 \parallel ... \parallel B_{31}, M$ **}**	
1.	$(M_L \parallel M_R) = M$
2.	$a = B_0, b = B_9, c = B_{16}, d = B_{19}, e = B_{30} \oplus M_L, f = B_{31} \oplus M_R$
3.	$(a', b', c', d', e', f') = F(a, b, c, d, e, f)$
4.	$B_0 = b', B_1 = c'$
5.	$B_i = B_{i-2}, 2 \leq i \leq 31$
6.	$M = M + 1$
7.	$k = a' \parallel e'$
Output = { $k, B_0 \parallel ... \parallel B_{31}, M$ **}**	

Table 3. Dragon's Keystream Generation Function

2.3 Keystream Generation

Dragon has a large NLFSR of 1024 bits divided into thirty two 32-bit words $B_i, 0 \leq i \leq 31$. During each round, six words from the internal state are used as inputs to the F function. The indices to these words are 0, 9, 16, 19, 30, 31, and

form a Full Positive Difference Set (FPDS). Additionally, a 64-bit memory component, M, acts as a counter in keystream generation, with the initial value for keystream generation being the final value of M defined by the key initialisation process. Each round of the keystream generation results in the output of a 64-bit word k. Table 3 shows one round of keystream generation. Note the output of the process is a keystream word k, and an updated state B and memory M.

3 Design Principles of Dragon

3.1 Design of F Function

All the keystream words and feedback words are dependent on all input words, both at the bit level and word level. A single bit change in any of the six input words results in completely different keystream and feedback words.

3.2 Design of S-boxes

Dragon uses two 8×32 S-boxes that have been designed heuristically to satisfy a range of important security related properties. They are used to create two nonlinear 32×32 mappings G and H. The simple construction (shown in Section 2.1) allows the non-linearity properties of the output bits of G and H to be calculated exactly from the known properties of the components of the underlying 8×32 S-boxes, S_1 and S_2. Both s-boxes were designed to have balanced component Boolean functions with:

- best known non-linearity of 116,
- optimum algebraic degree 6 or 7 according to Siegenthaler's tradeoff [18],
- low autocorrelation,
- distinct equivalence classes,
- all XOR pairs satisfying:
  - better than random non-linearity with 102 minimum,
  - almost balanced (the imbalance is not more than 16),
  - distinct equivalence classes,
  - same optimal degree as the components.

We adopt a standard notation (n, t, d, x, y) to describe Boolean function properties where n is the number of variables, t is the order of resiliency (where $t = 0$ indicates a balanced function), d is the algebraic degree, x is the non-linearity and y is the largest magnitude in the autocorrelation function. All the components of S_1 are $(8, 1, 6, 116, y)$ where $32 \leq y \leq 48$ which is considered sufficiently low. S_1 functions achieve the highest non-linearity possible for resilient functions. All the components of S_2 are $(8, 0, 7, 116, 24)$, where we note that the achieved autocorrelation of 24 is the lowest known for balanced functions of this size.

These s-boxes were created one output bit at a time using heuristic techniques. Existing methods [14] were adopted to generate the individual functions, then they were compared to the existing s-box functions to check the above-listed

requirements for the XOR pairs. When the candidate function was acceptable it was appended to the s-box, else another function was tested. We found it was possible to generate 32 functions for each s-box while satisfying the stringent requirements outlined above.

Finally we remark that the output functions of resulting Dragon virtual s-boxes G and H have higher non-linearity (at 116) than other popular 32×32 cryptographic mappings, such as the `SBOX/MIXCOL` operation from AES [15] and Mugi[19], which use only 112 non-linearity functions. Also Dragon's s-boxes avoid the linear redundancy weakness that is intrinsic to finite field operation based s-boxes [9] which are used in the international standard ciphers AES [15], Camellia [2].

3.3 Design of Key Initialisation

The key setup and keystream generation of Dragon both use the F function, for ease of implementation and efficiency. However, the key setup of Dragon is deliberately designed to be different to keystream generation, so that the mapping of internal state to the feedback is different.

There are three differences between the key setup and the keystream generation: the use of the 64-bit memory M, the size of the feedback and the FPDS selection used.

From Section 3.1, F is a reversible mapping, and the design of the key setup network uses this property of F to produce a bijective process. For any unique pair of K and IV, the key setup procedure initialises the internal state and M to unique values. Note that M is used as memory in key setup, but as a counter in keystream generation.

The feedback of Dragon consists of four words of the F function outputs, totalling 128 bits in key setup (the feedback size is 64 bits in keystream generation). This means that F can mix K and IV effectively in a minimum number of rounds. A smaller number of rounds in key setup translate directly into high rekeying performance. This makes Dragon very competitive in practical applications that require frequent rekeying, such as mobile and wireless transmissions that usually use the frame number as the IV.

A different FPDS is chosen for the key setup because of the change in the size of the feedback. The taps from the internal state, {0,4,6,7}, form a FPDS both in the forward and reverse direction. This is designed to frustrate the cryptanalysis of key setup by guess and determine techniques.

Dragon-128 and Dragon-256 are designed to have very similar initialisation process so that the speed is identical. However, another important design consideration is the use 128-bit and 256-bit key and initialisation vector pairs. We ensured that no pair of 256-bit K and IV can initialise Dragon to the same state as any arbitrary pair of 128-bit k and iv. This avoids the cryptanalyst reducing the search space in a brute force attack from 256-bit to 128-bit.

4 Analysis of Cipher

4.1 Statistical Tests

Statistical tests provided by the CRYPT-X [8] package were performed on keystream produced by the Dragon cipher. The frequency, binary derivative, change point, subblock and runs tests were executed with 30 streams of Dragon output, each eight megabits in length. The sequence and linear complexity tests were executed for the 30 streams with two hundred kilobits each. Dragon passed all pertinent statistical tests.

4.2 Period Length

Given that Dragon has a 1024-bit internal state, the expected period of the internal state is 2^{512}, assuming the mapping is pseudo-random [4]. For cryptographic use, establishing the lower bound for the period of the output sequence is critical. Each round of Dragon is under the influence of a 64-bit counter, M. Since the counter M has a period of 2^{64}, the period of Dragon's internal state is lower bounded by 2^{64}. Taken together, the internal state and the counter M give Dragon an expected period of 2^{576}.

The amount of keystream produced by a unique pair of K and IV is limited to 2^{64} bits (in most applications the actual keystream would be much smaller) in the specification of Dragon. This is a small fraction of the lower bound of the period (and a very small fraction of the expected period), and therefore avoids the possibility of keystream collision attacks.

4.3 Weak Keys

Weak keys are those keys that bypass some operations of the cipher. That is, the operations have no effect in the calculation of the feedback or the output keystream.

Dragon is designed to avoid weak keys. The internal state is an NLFSR, therefore the all zero state is not a problem as Dragon is designed to avoid fixed points. While it is easy to bypass the pre-mixing phase in a single iteration of the F function by having repetitive inputs such as all zeros or all ones, it is only possible for the first of the 16 iterations of F in the key setup. Also, selected values are limited to the first four inputs of the F function, as the last two inputs take the value of M. The network of G and H functions ensure that the initial states which bypass the pre-mixing phase cannot bypass any other operations in F. We believe that the above design features provide a strong guarantee that there are no weak keys for Dragon.

5 Cryptanalysis of Dragon

5.1 Related Key and IV Attacks

The Dragon rekeying strategy is simple, and the use of initialisation vectors provides a way to reuse a master key without generating identical keystreams.

The rekeying strategy prevents related key and IV attacks before even the first word of output is generated, by mixing each bit of the key into all words of the initial state over 16 rounds of the highly non-linear F function defined for the keystream generator module. This function has six 32-bit inputs and six 32-bit outputs. During rekeying, the F function is iterated 16 times, each time populating the leftmost side of the internal state with 128 bits comprising four outputs. After eight rounds, all of the initial keying material in the state has been replaced by unknown output from the F function.

Of the six inputs to the F function, four words are taken directly from the keyed internal state, while two are taken from a 64-bit memory M. The contents of this memory are initially known, since they are determined by a published constant. Also, the memory can not be manipulated by the attacker in the same way as the internal state, since it is not keyed. Two outputs from the F function feedback to the memory, making its value hard to determine after the first round. All output words of F are affected by the memory, increasing the difficulty that the attacker faces in controlling inputs to subsequent rounds.

Diffusion One strategy in an attack is to minimise the number of words with a non-zero difference in the internal state. The aim of this strategy is controllability. The larger the number of non-zero words used as input to the non-linear function, the more complex the resulting output. The key schedule of Dragon is designed so that after 12 rounds, even an initial difference of single word difference is propagated to all words in the internal state (see Table 4). Since there are 16 rounds, this is an ample margin to ensure an attacker is unable to determine the state contents after rekeying. The speed of this diffusion is aided by the fact that the first word of the state is used as input to F function, and the output of the F function replaces the first word.

1	0	ΔA	0	0	0	0	0	0
2	0	0	ΔA	0	0	0	0	0
3	0	0	0	ΔA	0	0	0	0
4	0	0	0	0	ΔA	0	0	0
5	ΔA	0	0	0	0	ΔA	0	0
6	ΔB	ΔA	0	0	0	0	ΔA	0
7	ΔC	ΔB	ΔA	0	0	0	0	ΔA
8	ΔD	ΔC	ΔB	ΔA	0	0	0	0
9	ΔE	ΔD	ΔC	ΔB	ΔA	0	0	0
10	ΔF	ΔE	ΔD	ΔC	ΔB	ΔA	0	0
11	ΔG	ΔF	ΔE	ΔD	ΔC	ΔB	ΔA	0
12	ΔH	ΔG	ΔF	ΔE	ΔD	ΔC	ΔB	ΔA

Table 4. Propagation of non-zero difference in internal state of the rekeying

Even a single round of Dragon F function prevents high probability differentials due to its use of the G and H functions, and high diffusion. A single input difference is propagated to differences in each of the outputs. The F function consists of three layers: pre-mixing, confusion through s-box application, and post-mixing. Referring to the notation of section 2.1, only inputs a, b, c and d can be initially and indirectly controlled by an attacker, since e and f come from internal and inaccessible memory.

The attacker may wish to make use of the fact that b and d are mixed with only one other word in the pre-mixing phase, while a and c are mixed with two others. For the input $-(e \oplus f), b, -(b \oplus e \oplus f), -(b \oplus e \oplus f), e, e \oplus f)$ the pre-mixing stage produces the output $(0, b \oplus -(e \oplus f), 0, 0, e, e \oplus f)$. For difference input Δb, this produces the difference $(0, \Delta b, 0, 0, 0, 0)$ since e and f are at this stage constants. This bypasses the G row of s-boxes and activates a single s-box in the second row to produce the post-mixing input $(\Delta H_1(\Delta b), \Delta b, 0, 0, 0, 0)$. The post-mixing output is $(\Delta H_1(\Delta b), \Delta b, \Delta b, (\Delta H_1(\Delta b), 0, 0))$.

At this stage, all of the feedback words to the internal state are non-zero. However, the difference of the feedback to the internal state is still zero. This fact cannot be exploited by the attacker since the input differences to this round are not reproducible in later rounds, and thus the difference of the internal memory cannot be maintained. Consequently Dragon is not vulnerable to related key attacks that are more efficient than a brute force search of the 256-bit key.

5.2 Time-Memory Tradeoff Attacks

Time-Memory tradeoff attacks [3] rely on pre-computation to reduce the effort required for a key recovery attack on a keystream. The attack comprises two steps. The first, the preprocessing step, sees the attacker calculating a table of keys or internal states and corresponding keystream prefixes. The table is ordered upon the prefix. The second step involves observing keystreams and attempting to match each against a prefix in the table. If the match is successful, then with some likelihood the internal state is known by reading the opposing entry in the table.

The parameters in an attack are time (T), memory (M), and amount of data (D). Generally, $T \times M^2 \times D^2 = S^2$ where S is the state space of the cipher, and $D^2 \leq T$ [3]. The pre-computation time P is equal to $S \div D$.

Dragon has an internal state space of 1088 bits (including the 64-bit memory). Since the design strength of Dragon is 256 bits, the time-memory tradeoff attack is infeasible. For the brute-force equivalent attack with $T = 2^{256}$, data requirements are limited to 2^{64} bits, which imposes a lower bound on memory for the attack of 2^{896} bits.

5.3 Guess and Determine Attacks

The indices {0, 9, 16, 19, 30, 31} of the state elements used in Dragon's update function form a full positive difference set. This is a design decision to prevent guess and determine attacks [10].

In keystream generation, guessing six inputs (192 bits) to F in a round allows an attacker to calculate the feedback words b' and c' and the keystream words a' and e', which can be used to discard most incorrect guesses. At this point the attacker has knowledge of the state words at indices {0, 1, 10, 17, 20} and some information about the value of B_{31} and M. However, the FPDS selection of the internal state means that to obtain the next pair of keystream words, guessing a further five inputs (160 bits) is necessary. The attacker can attempt to jump ahead to a future keystream word pair, but again the FPDS means that the attacker needs to guess five inputs. This rapid increase in the number of possible guess pathways makes the attack infeasible. In addition, the interplay of B_{30}, B_{31} and M means there will be more than one set of values for these three elements for a unique pair of e and f, further complicating the cryptanalytic attempt by guess and determine.

The attacker is unable to reduce the complexity of a guess and determine attack by guessing individual state bytes, rather than whole words. The use of large s-boxes (G and H functions are effectively 32×32 s-boxes) means that guessing three of the four input bytes is insufficient to deduce any byte of the s-box output.

To calculate keystream words from two rounds of Dragon, the attacker is required to guess more than 256 bits of the internal state. This is worse than exhaustive key search, and makes guess and determine attacks on Dragon infeasible.

5.4 Distinguishing Attacks

If the output sequence of a stream cipher can be statistically distinguished from a random sequence, then the cipher is not strong enough for cryptographic applications. Dragon is designed with a large state and complex initialisation and update function. It has no linear masking, and therefore immune to this type of distinguishing attacks [5]. It is expected to have a very large period of 2^{582} (with a lower bound of 2^{64} because of the influence of the 64-bit counter) and it passes standard statistically tests for randomness. The amount of keystream output for an unique pair key and initialisation vector is limited to 2^{64} bits. We conjecture that it is impractical to collect an amount of output sufficient to distinguish Dragon keystream output from a random binary sequence.

5.5 Linear Approximations

From Lemma 15 of [17], the non-linearity of the sum of disjoint functions can be calculated as follows. Let $g(x_1, \ldots, x_s, y_1, \ldots, y_t) = f_1(x_1, \ldots, x_s) \oplus f_2(y_1, \ldots, y_t)$. Then the non-linearity of g satisfies $N_g \geq 2^{s+t-1} - \frac{1}{2} P_1 \cdot P_2$ where P_1 and P_2 are the maximum Walsh-Hadamard transform values of f_1 and f_2, respectively.

The G and H functions of Dragon are composed from two 8×32 s-boxes, S_1 and S_2. Both s-boxes have all outputs with non-linearity 116, therefore $P_{S_1} = P_{S_2} = 2^8 - 2 \cdot 116 = 24$. The non-linearity of the output bits of the G and H functions can then be calculated as $N_G = N_H \geq 2^{8+8+8+8-1} - \frac{1}{2} \cdot 24 \cdot 24 \cdot 24 \cdot 24 =$

$2^{31} - 165888$. The best affine approximation to the G or H function output bits has bias no greater than $\frac{2^{31}-2^{31}+165888}{2^{31}} = 2^{-14.66}$. At any given round, the keystream words of Dragon are the results of five G or H functions each, hence the best affine approximation to the Dragon F function output bits has bias no greater than $(2^{-14.66})^5 = 2^{-73.3}$.

Linear cryptanalysis requires equations relating the key bits to the internal state bits, and in turn the keystream bits, where the internal state variables can be cancelled. The complete mixing of Dragon's key setup avoids the divide and conquer approach, therefore all the internal state variables are needed in the linear equations. The output keystream will be dependent on all 1,024 bits of the initial internal state after 8 iterations of F. The bias of the best affine approximation over 8 iterations of F is no greater than $(2^{-73.3})^8 = 2^{-586.4}$. As the key size of Dragon is 256 bits, attack on Dragon using linear approximation has complexity greater than exhaustive key search.

5.6 Algebraic Attacks

Successful algebraic attacks on keystream generators [6] have so far been restricted mainly to LFSR based generators. The general attack model consists of the internal state S, the linear update function L and the output function f. Let S_0 denote the internal state at time $t = 0$, and $L^t(S_0)$ denote the internal state at time t. The attacker constructs a system of equations relating the internal state bits with the observed keystream bits, where $z_t = f(L^t(S_0))$ at time t. The attacker can set up a large number of equations just by merely collecting keystream bits, since the internal state at time t can easily be derived from the linear nature of LFSRs.

This model cannot be applied to Dragon since the update function is non-linear. Let the non-linear update function be N, then the equation becomes $z_t = f(N^t(S_0))$. Note that N has a poor linear approximation of $2^{-73.3}$ as shown in Section 5.5. The lack of the linear update function means the attacker can not simply calculate the internal state at time t to construct the system of equations.

When constructing the system of equations for Dragon, the degree of equations would grow exponentially. This is easy to see as any output of G or H is a degree 7 function of the inputs since S_2 has algebraic order 7. If we approximate $\boxplus$ with $\oplus$, we can then write equations of degree $7^2 = 49$ that maps the 192 input bits to the first 64 output keystream bits. However, the feedback is used immediately in the production of the next 64 bits of keystream, and results in equations of degree $7^4 = 2,401$. Note that at this point, the inputs consist of only 352 bits, and therefore the equations would be limited to degree 352. The degree of the equations would grow to the full 1024 bits of the internal state, after 8 iterations of the F function, or 512 bits of keystream produced.

Using the technique published in [7] to describe the 8×32 s-boxes of Dragon using quadratic equations results in 565 quadratic equations in 256 monomials for each s-box (identical to the analysis of CAST [1]). Again, let us approximate $\boxplus$ with $\oplus$, then after 8 iterations of F, the system of equations has degree 1,024

as well. This is to say, even if there existed some annihilators [13] that reduce Dragon's Boolean functions right down to quadratic, the degree of the overall equations would still grow to unmanageable sizes.

It is clear that the system of equations for Dragon will be very difficult to solve, if it is solvable at all. Furthermore, it will require far more effort than exhaustive key search since solving techniques all have complexities exponential in the degree of the equations. It is interesting to note that the above analysis approximates modular addition with XOR, and thus resulting in a weaker version of Dragon. With the modular addition in place, it will be even more difficult for algebraic attacks to succeed against Dragon (see similar example of the effect of modular addition in CAST [1]).

6 Implementation and Performance

Dragon is designed to be efficient in both software and hardware, in terms of throughput and a small implementation footprint. Its 32-bit word size is chosen to match that of the ubiquitous Intel Pentium family, since this leads to the best software efficiency on that platform. Note that the results presented in this Section apply to both Dragon-128 and Dragon-256.

6.1 Software

On an Intel Pentium 4, a naïve C implementation of Dragon produces one byte of keystream every 6.74 clock cycles, and 1,395 cycles per rekeying operation. On a 3.2GHz Pentium 4, the throughput of Dragon is 3.8Gbps. This is competitive with many of its peers, including SNOW 2 (5.5 cycles/byte), Turing (6.1 cycles/byte) and RC4 (7.1 cycles/byte).

Storage requirements include 2,048 bytes to store Dragon's two 8×32 s-boxes, 1,024 bits (128 bytes) for the internal state, and a further 8 bytes for the 64-bit counter. Including temporary variables and an object code size of 2,810 bytes, Dragon has memory requirements totalling 4,994 bytes. This is suitable for even very constrained environments.

A reference implementation of Dragon written in C can be obtained from `http://www.isrc.qut.edu.au/resource/dragon/`.

6.2 Hardware

The design of Dragon allows high degree of parallelisation in hardware. The operations on the six inputs of the F function can be divided into three groups, each operating on two inputs. The pre-mixing and the post-mixing are implemented using 32-bit modular adders. The G and H functions are implemented using look-up tables and XOR operations. The hardware complexity is about 6,524 gates and 196,672 bits of memory. On Samsung 0.13um ASIC running at 2.6GHz, the minimum delay is 2.774ns with a throughput of 23Gbps.

The speed in hardware can be improved by using m-parallel-structure proposed in [12]. This hardware implementation strategy applies to all shift registers, and achieves an m times increase in efficiency with m times increase in hardware complexity. On Altera FPGA/CPLD running at 16.67MHz, an implementation of Dragon achieves a throughput of 1.06Gbps with 16 times hardware complexity.

7 Conclusion

This paper presents Dragon, a new stream cipher constructed upon a word based non-linear feedback shift register. The key and initialisation vector are 128 bits for Dragon-128 and 256 bits for Dragon-256. Dragon is designed with both security and efficiency in mind. It has been shown that Dragon is secure against all known cryptanalytic attacks.

References

1. C. Adams. Designing Against the 'Overdefined System of Equations' Attack, May 2004. `http://eprint.iacr.org/2004/110/`.
2. K. Aoki, T. Ichikawa, M. Kanda, M. Matsui, S. Moriai, J. Nakajima, and T. Tokita. Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In D. Stinson and S. Tavares, editors, *Selected Areas in Cryptography: 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 14-15, 2000. Proceedings*, volume 2012 of *Lecture Notes in Computer Science*, pages 39–56, Berlin Heidelberg, 2000. Springer-Verlag.
3. A. Biryukov and A. Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In T. Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000. Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13, Berlin Heidelberg, 2000. Springer-Verlag.
4. W. Chambers. On Random Mappings and Random Permutations. In B. Preneel, editor, *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994. Proceedings*, volume 1008 of *Lecture Notes in Computer Science*, pages 22–28, Berlin Heidelberg, 1995. Springer-Verlag.
5. D. Coppersmith, S. Halevi, and C. Jutla. Cryptanalysis of Stream Ciphers with Linear Masking. In M. Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 515–532, Berlin Heidelberg, 2002. Springer-Verlag.
6. N. Courtois. Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt. In P. Lee and C. Lim, editors, *Information Security and Cryptology - ICISC 2002, 5th International Conference Seoul, Korea, November 28-29, 2002, Revised Papers*, volume 2587 of *Lecture Notes in Computer Science*, pages 182–199, Berlin Heidelberg, 2003. Springer-Verlag.
7. N. Courtois and J. Pieprzyk. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations. In Y. Zheng, editor, *Advances in Cryptology - ASIACRYPT*

2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002. Proceedings, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287, Berlin Heidelberg, 2002. Springer-Verlag.

8. E. Dawson, A. Clark, G. Gustafson, and L. May. *CRYPT-X'98 User Manual*, 1999.
9. J. Fuller and W. Millan. Linear Redundancy in S-Boxes. In T. Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003. Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 74–86, Berlin Heidelberg, 2003. Springer-Verlag.
10. P. Hawkes and G. Rose. Guess-and-Determine Attacks on SNOW. In K. Nyberg and H. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 37–46, Berlin Heidelberg, 2003. Springer-Verlag.
11. J. Kam and G. Davida. Structured Design of Substitution-Permutation Encryption Networks. *IEEE Transactions on Computers*, 28(10):747–753, October 1979.
12. H. Lee and S. Moon. Parallel Stream Cipher for Secure High-Speed Communications. *Signal Processing*, 82(2):137–143, February 2002.
13. W. Meier, E. Pasalic, and C. Carlet. Algebraic Attacks and Decomposition of Boolean Functions. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 474–491, Berlin Heidelberg, 2004. Springer-Verlag.
14. W. Millan, J. Fuller, and E. Dawson. New Concepts in Evolutionary Search for Boolean Functions in Cryptology. In *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 3, pages 2157–2164. IEEE, 2003.
15. National Institute of Standards and Technology. Federal Information Processing Standards Publication 197, 2001.
16. G. Rose and P. Hawkes. Turing: A Fast Stream Cipher. In T. Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003. Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 290–306, Berlin Heidelberg, 2003. Springer-Verlag.
17. J. Seberry, X. Zhang, and Y. Zheng. Nonlinearly Balanced Boolean Functions and Their Propagation Characteristics. In D. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993. Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 49–60, Berlin Heidelberg, 1994. Springer-Verlag.
18. T. Siegenthaler. Correlation Immunity of Nonlinear Combining Functions for Cryptographic Applications. *IEEE Transactions on Information Theory*, 30(5):776–780, September 1984.
19. D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi, and B. Preneel. A New Keystream Generator MUGI. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption, 9th International Workshop, FSE 2002, Leuven, Belgium, February 4-6, 2002. Revised Papers*, volume 2365 of *Lecture Notes in Computer Science*, pages 179–194, Berlin Heidelberg, 2002. Springer-Verlag.

A Test Vectors

128-BIT KEY AND IV

KEY:
00001111 22223333 44445555 66667777
IV:
00001111 22223333 44445555 66667777
KEYSTREAM:
99B3AA14 B63BD02F E14358A4 54950425 F4B0D3FD 8BA69178 E0392938 A718C165
2E3BEB1E 11613D58 9EABB9F5 43A1C51C 73C1F227 9D1CAEA8 5C55F539 BAFD3C59
ECAC88BD 17EB1C9D A28DD63E 9093C913 3032D918 3A9B33BC 2933A79D 75669827
20EF3004 C53B0253 7A1BE796 29F8D9A3 8DC1FD31 ED9D1100 B07DFFB1 AC75EB31

KEY:
00112233 44556677 8899AABB CCDDEEFF
IV:
00112233 44556677 8899AABB CCDDEEFF
KEYSTREAM:
98821506 0E87E695 EB7AEF36 313FF910 E6C7312F 30357424 4922043D 98146EE2
202D4D49 6C602ECC 937DD3F4 E39BE26C 849DB415 F04C540E 88588C7A A3C65A31
E2156229 1E86028B 3F5A21B9 4A94C135 B3A01527 747E6521 FFEE14F0 FA1FCC73
74C8B204 4009F57D 1D63007E F1D8D221 E429EBA8 60F56098 45891D74 716694B2

256-BIT KEY AND IV

KEY:
00001111 22223333 44445555 66667777 88889999 AAAABBBB CCCCDDDD EEEEFFFF
IV:
00001111 22223333 44445555 66667777 88889999 AAAABBBB CCCCDDDD EEEEFFFF
KEYSTREAM:
BC020767 DC48DAE3 14778D8C 927E8B32 E086C6CD E593C008 600C9D47 A488F622
3A2B94D6 B853D644 27E93362 ABB8BA21 751CAAF7 BD316595 2A37FC1E A3F12FE2
5C133BA7 4C15CE4B 3542FDF8 93DAA751 F5710256 49795D54 31914EBA 0DE2C2A7
8013D29B 56D4A028 3EB6F312 7644ECFE 38B9CA11 1924FBC9 4A0A30F2 AFFF5FE0

KEY:
00112233 44556677 8899AABB CCDDEEFF 00112233 44556677 8899AABB CCDDEEFF
IV:
00112233 44556677 8899AABB CCDDEEFF 00112233 44556677 8899AABB CCDDEEFF
KEYSTREAM:
8D3AB9BA 01DAA3EB 5CBD0F6D E3ECFCAB 619AF808 CF9C4A42 E2877766 6D2D7037
EE6F94AC 29D1EEE5 340DB047 8E91A679 480D8D88 2367CE2A 31C96AD4 49E70756
815EBEB2 290DBA7A 3CCB76A2 257BD122 2B0B7AED 917FAFFF 6B58B2B2 B05F24F6
E271A016 9E897BEF F5C22451 DA6F9E40 52B78BE5 6C97C1A5 C6F8E791 0F7B9C98

B Dragon's S-Boxes

```
sbox1[256]={
 0x393BCE6B,0x232BA00D,0x84E18ADA,0x84557BA7,0x56828948,0x166908F3,
 0x414A3437,0x7BB44897,0x2315BE89,0x7A01F224,0x7056AA5D,0x121A3917,
 0xE3F47FA2,0x1F99D0AD,0x9BAD518B,0x99B9E75F,0x8829A7ED,0x2C511CA9,
 0x1D89BF75,0xF2F8CDD0,0x2DA2C498,0x48314C42,0x922D9AF6,0xAA6CE00C,
 0xAC66E078,0x7D4CB0C0,0x5500C6E8,0x23E4576B,0x6B365D40,0xEE171139,
 0x336BE860,0x5DBEEEFE,0x0E945776,0xD4D52CC4,0x0E9BB490,0x376EB6FD,
 0x6D891655,0xD4078FEE,0xE07401E7,0xA1E4350C,0xABC78246,0x73409C02,
 0x24704A1F,0x478ABB2C,0xA0849634,0x9E9E5FEB,0x77363D8D,0xD350BC21,
 0x876E1BB5,0xC8F55C9D,0xD112F39F,0xDF1A0245,0x9711B3F0,0xA3534F64,
 0x42FB629E,0x15EAD26A,0xD1CFA296,0x7B445FEE,0x88C28D4A,0xCA6A8992,
 0xB40726AB,0x508C65BC,0xBE87B3B9,0x4A894942,0x9AEECC5B,0x6CA6F10B,
 0x303F8934,0xD7A8693A,0x7C8A16E4,0xB8CF0AC9,0xAD14B784,0x819FF9F0,
 0xF20DCDFA,0xB7CB7159,0x58F3199F,0x9855E43B,0x1DF6C2D6,0x46114185,
 0xE46F5D0F,0xAAC70B5B,0x48590537,0x0FD77B28,0x67D16C70,0x75AE53F4,
 0xF7BFECA1,0x6017B2D2,0xD8A0FA28,0xB8FC2E0D,0x80168E15,0x0D7DEC9D,
 0xC5581F55,0xBE4A2783,0xD27012FE,0x53EA81CA,0xEBAA07D2,0x54F5D41D,
 0xABB26FA6,0x41B9EAD9,0xA48174C7,0x1F3026F0,0xEFBADD8E,0x387E9014,
 0x1505AB79,0xEADF0DF7,0x67755401,0xDA2EF962,0x41670B0E,0x0E8642F2,
 0xCE486070,0xA47D3312,0x4D7343A7,0xECDA58D0,0x1F79D536,0xD362576B,
 0x9D3A6023,0xC795A610,0xAE4DF639,0x60C0B14E,0xC6DD8E02,0xBDE93F4E,
 0xB7C3B0FF,0x2BE6BCAD,0xE4B3FDFD,0x79897325,0x3038798B,0x08AE6353,
 0x7D1D20EB,0x3B208D21,0xD0D6D104,0xC5244327,0x9893F59F,0xE976832A,
 0xB1EB320B,0xA409D915,0x7EC6B543,0x66E54F98,0x5FF805DC,0x599B223F,
 0xAD78B682,0x2CF5C6E8,0x4FC71D63,0x08F8FED1,0x81C3C49A,0xE4D0A778,
 0xB5D369CC,0x2DA336BE,0x76BC87CB,0x957A1878,0xFA136FBA,0x8F3C0E7B,
 0x7A1FF157,0x598324AE,0xFFBAAC22,0xD67DE9E6,0x3EB52897,0x4E07E855,
 0x87CE73F5,0x8D046706,0xD42D18F2,0xE71B1727,0x38473B38,0xB37B24D5,
 0x381C6AE1,0xE77D6589,0x6018CBFF,0x93CF3752,0x9B6EA235,0x504A50E8,
 0x464EA180,0x86AFBE5E,0xCC2D6AB0,0xAB91707B,0x1DB4D579,0xF9FAFD24,
 0x2B28CC54,0xCDCFD6B3,0x68A30978,0x43A6DFD7,0xC81DD98E,0xA6C2FD31,
 0x0FD07543,0xAFB400CC,0x5AF11A03,0x2647A909,0x24791387,0x5CFB4802,
 0x88CE4D29,0x353F5F5E,0x7038F851,0xF1F1C0AF,0x78EC6335,0xF2201AD1,
 0xDF403561,0x4462DFC7,0xE22C5044,0x9C829EA3,0x43FD6EAE,0x7A42B3A7,
 0x5BFAAAEC,0x3E046853,0x5789D266,0xE1219370,0xB2C420F8,0x3218BD4E,
 0x84590D94,0xD51D3A8C,0xA3AB3D24,0x2A339E3D,0xFEE67A23,0xAF844391,
 0x17465609,0xA99AD0A1,0x05CA597B,0x6024A656,0x0BF05203,0x8F559DDC,
 0x894A1911,0x909F21B4,0x6A7B63CE,0xE28DD7E7,0x4178AA3D,0x4346A7AA,
 0xA1845E4C,0x166735F4,0x639CA159,0x58940419,0x4E4F177A,0xD17959B2,
 0x12AA6FFD,0x1D39A8BE,0x7667F5AC,0xED0CE165,0xF1658FD8,0x28B04E02,
 0x1FA480CF,0xD3FB6FEF,0xED336CCB,0x9EE3CA39,0x9F224202,0x2D12D6E8,
 0xFAAC50CE,0xFA1E98AE,0x61498532,0x03678CC0,0x9E85EFD7,0x3069CE1A,
 0xF115D008,0x4553AA9F,0x3194BE09,0xB4A9367D,0x0A9DFEEC,0x7CA002D6,
 0x8E53A875,0x965E8183,0x14D79DAC,0x0192B555};
```

```
sbox2[256]={
 0xA94BC384,0xF7A81CAE,0xAB84ECD4,0x00DEF340,0x8E2329B8,0x23AF3A22,
 0x23C241FA,0xAED8729E,0x2E59357F,0xC3ED78AB,0x687724BB,0x7663886F,
 0x1669AA35,0x5966EAC1,0xD574C543,0xDBC3F2FF,0x4DD44303,0xCD4F8D01,
 0x0CBF1D6F,0xA8169D59,0x87841E00,0x3C515AD4,0x708784D6,0x13EB675F,
 0x57592B96,0x07836744,0x3E721D90,0x26DAA84F,0x253A4E4D,0xE4FA37D5,
 0x9C0830E4,0xD7F20466,0xD41745BD,0x1275129B,0x33D0F724,0xE234C68A,
 0x4CA1F260,0x2BB0B2B6,0xBD543A87,0x4ABD3789,0x87A84A81,0x948104EB,
 0xA9AAC3EA,0xBAC5B4FE,0xD4479EB6,0xC4108568,0xE144693B,0x5760C117,
 0x48A9A1A6,0xA987B887,0xDF7C74E0,0xBC0682D7,0xEDB7705D,0x57BFFEAA,
 0x8A0BD4F1,0x1A98D448,0xEA4615C9,0x99E0CBD6,0x780E39A3,0xADBCD406,
 0x84DA1362,0x7A0E984B,0xBED853E6,0xD05D610B,0x9CAC6A28,0x1682ACDF,
 0x889F605F,0x9EE2FEBA,0xDB556C92,0x86818021,0x3CC5BEA1,0x75A934C6,
 0x95574478,0x31A92B9B,0xBFE3E92B,0xB28067AE,0xD862D848,0x0732A22D,
 0x840EF879,0x79FFA920,0x0124C8BB,0x26C75B69,0xC3DAAAC5,0x6E71F2E9,
 0x9FD4AFA6,0x474D0702,0x8B6AD73E,0xF5714E20,0xE608A352,0x2BF644F8,
 0x4DF9A8BC,0xB71EAD7E,0x6335F5FB,0x0A271CE3,0xD2B552BB,0x3834A0C3,
 0x341C5908,0x0674A87B,0x8C87C0F1,0xFF0842FC,0x48C46BDB,0x30826DF8,
 0x8B82CE8E,0x0235C905,0xDE4844C3,0x296DF078,0xEFAA6FEA,0x6CB98D67,
 0x6E959632,0xD5D3732F,0x68D95F19,0x43FC0148,0xF808C7B1,0xD45DBD5D,
 0x5DD1B83B,0x8BA824FD,0xC0449E98,0xB743CC56,0x41FADDAC,0x141E9B1C,
 0x8B937233,0x9B59DCA7,0xF1C871AD,0x6C678B4D,0x46617752,0xAAE49354,
 0xCABE8156,0x6D0AC54C,0x680CA74C,0x5CD82B3F,0xA1C72A59,0x336EFB54,
 0xD3B1A748,0xF4EB40D5,0x0ADB36CF,0x59FA1CE0,0x2C694FF9,0x5CE2F81A,
 0x469B9E34,0xCE74A493,0x08B55111,0xEDED517C,0x1695D6FE,0xE37C7EC7,
 0x57827B93,0x0E02A748,0x6E4A9C0F,0x4D840764,0x9DFFC45C,0x891D29D7,
 0xF9AD0D52,0x3F663F69,0xD00A91B9,0x615E2398,0xEDBBC423,0x09397968,
 0xE42D6B68,0x24C7EFB1,0x384D472C,0x3F0CE39F,0xD02E9787,0xC326F415,
 0x9E135320,0x150CB9E2,0xED94AFC7,0x236EAB0F,0x596807A0,0x0BD61C36,
 0xA29E8F57,0x0D8099A5,0x520200EA,0xD11FF96C,0x5FF47467,0x575C0B39,
 0x0FC89690,0xB1FBACE8,0x7A957D16,0xB54D9F76,0x21DC77FB,0x6DE85CF5,
 0xBFE7AEE9,0xC49571A9,0x7F1DE4DA,0x29E03484,0x786BA455,0xC26E2109,
 0x4A0215F4,0x44BFF99C,0x711A2414,0xFDE9CDD0,0xDCE15B77,0x66D37887,
 0xF006CB92,0x27429119,0xF37B9784,0x9BE182D9,0xF21B8C34,0x732CAD2D,
 0xAF8A6A60,0x33A5D3AF,0x633E2688,0x5EAB5FD1,0x23E6017A,0xAC27A7CF,
 0xF0FC5A0E,0xCC857A5D,0x20FB7B56,0x3241F4CD,0xE132B8F7,0x4BB37056,
 0xDA1D5F94,0x76E08321,0xE1936A9C,0x876C99C3,0x2B8A5877,0xEB6E3836,
 0x9ED8A201,0xB49B5122,0xB1199638,0xA0A4AF2B,0x15F50A42,0x775F3759,
 0x41291099,0xB6131D94,0x9A563075,0x224D1EB1,0x12BB0FA2,0xFF9BFC8C,
 0x58237F23,0x98EF2A15,0xD6BCCF8A,0xB340DC66,0x0D7743F0,0x13372812,
 0x6279F82B,0x4E45E519,0x98B4BE06,0x71375BAE,0x2173ED47,0x14148267,
 0xB7AB85B5,0xA875E314,0x1372F18D,0xFD105270,0xB83F161F,0x5C175260,
 0x44FFD49F,0xD428C4F6,0x2C2002FC,0xF2797BAF,0xA3B20A4E,0xB9BF1A89,
 0xE4ABA5E2,0xC912C58D,0x96516F9A,0x51561E77};
```