Morpheus: Stream Cipher for Software & Hardware Applications

Nikos Komninos
Algorithms & Security Group
Athens Information Technology
GR-190 02, Peania (Attiki), Greece
nkom@ait.edu.gr

ABSTRACT

In a world where electronic devices with different characteristics are networked, privacy is an essential element for the communicating process. Privacy can be achieved by encryption algorithms with unique features based on the application that are deployed. In this paper a word-oriented stream cipher, or Morpheus, for both hardware and software devices, is proposed. Morpheus targets multimedia applications, such as Games-On-Demand or IPTV, where data are usually streamed over different kind of networks and devices. Morpheus behaves very well in all known statistical tests and is resilient to known attacks for both synchronous and self-synchronous encryption modes.

I. INTRODUCTION

There is a big industry demand for secure and efficient stream ciphers, the mobile/wireless communication sector being one of the foremost "consumers" of such ciphers. Stream ciphers are considered "more" appropriate than block ciphers, and in some cases mandatory, when buffering is limited or when characters must be individually processed as they are received. Because they have limited or no error propagation, stream ciphers may also be advantageous in situations where transmissions errors are highly probable. From scientific point of view, there is also an interest to get a better understanding for how to design stream ciphers, since many of the proposed schemes in the past have been more or less severely attacked.

Nowadays, the European Network of Excellence for Cryptology (ECRYPT) funded within the Information Society Program (IST) has launched the eSTREAM project, a stream cipher contest whose purpose is to identify new stream ciphers that might become suitable for widespread adoption. The eSTREAM project has defined mainly two categories, or profiles in the project context, for software and/or hardware applications. Some have emphasized the importance of including an authentication method and so two further profiles have been proposed which combine authentication methods in stream ciphers for software and/or hardware applications.

Current stream ciphers, including those in eSTREAM project, do not provide high flexibility to software and hardware applications. In this paper we propose Morpheus, a stream cipher that can be efficiently used for both software and hardware applications since it uses simple mathematical operations, such as eXclusive-OR, addition modulo 2^w, lookup table and fixed rotations that can be implemented efficiently in both software and hardware. Several tests were performed to Morpheus in synchronous and self-synchronous encryption modes, i.e., when the generation of the keystream was both independent and dependent of the plaintext and ciphertext.

Following this introduction, this paper is organized as follows. Section II presents current efforts to the eSTREAM project with ciphers such as Phelix, LEX and Salsa20 suitable for hardware and/or software applications. Section III analyzes in detail the design of the Morpheus algorithm. Section IV discusses security, implementation, and cryptanalysis issues of the algorithm. This paper concludes with remarks and comments on the proposed algorithm.

II. RELATED WORK

Several stream cipher designs have been proposed that are efficient in either software or hardware or both. Here we present three relatively recent designs which introduced some interesting cryptographic techniques some of which where adopted in the design of Morpheus. All of them were submitted to the eSTREAM Project as being both hardware and/or software efficient and have successfully passed the first round of evaluation.

Whiting et al. [7] proposed the Phelix algorithm that has successfully passed the first evaluation phase and it is implemented together with a Message Authentication Code (MAC). Phelix design consists of a network of simple bitwise operations with an initial state of eight 32-bit words; four "active" words that participate in the block update function and four "old" words which are used in the keystream output function. The operations used in the network are bitwise exclusive-or, addition modulo 232 and fixed left cyclic shifts, or rotations. The generation of the MAC is conducted after the encryption of the plaintext, which consists of 12 Phelix rounds with specific parameters. The keystreams generated by the last 4 of these rounds represent the MAC tag. However, some weaknesses have been published by Wu and Preneel introducing some differential attacks against Phelix [9, 10].

Biryukov [3] proposed LEX stream cipher for software applications only and has successfully passed the first verification phase of the eSTREAM project. LEX introduces the notion of a leak extraction from the Advanced Encryption Standard (AES), which is used in a chain-like design. The keystream is generated by the intermediate rounds of AES; on each round, a certain part of the 16-byte state is leaked, which depends on whether the round is odd or even. According to the author, this cipher is 2.5 times faster than AES in the 128-bit key version, 3 times faster in the 192-bit version and 3.5

times in 256-bit versions. However, based on an attack in [11], LEX design was slightly modified in order to be qualified to the second phase of the eSTREAM project.

Salsa20, another stream cipher for software applications, was proposed by Bernstein [2] and has entered the second phase of eSTREAM in the software category. In fact, Salsa20 is a hash function with 64-byte input and 64-byte output and is used in counter mode as a stream cipher. The hash function is implemented by four mixing functions and an invertible function that transforms a 4-byte sequence into a 32-bit word in a little-endian manner.

The first of the mixing functions, called *quarterround*, mixes four 32-bit words, by using XORs, additions modulo 232 and left rotations, and produces four new 32-bit words. The next two functions, *rowround* and *columnround*, are very similar to each other. They both mix sixteen 32-bit words, by applying four times the *quarterround* function, and return sixteen new 32-bit words. Their difference is that, if the sixteen words where seen as a 4x4 matrix, then the *rowround* function modifies the rows and the *columnround* function modifies the columns of this matrix. The last mixing function, named *doubleround*, mixes sixteen 32-bit words by applying first the *rowround* and next the *columnround* function. Although several attacks on Salsa20 were published, the cipher was left unchanged by its author.

III. An Overview of Morpheus

Morpheus is a word-oriented stream cipher designed to use a 128, 256, and 512 bits key and a 128-bit initialization vector (IV). The key is secret, and the IV is typically public knowledge. It produces a keystream, which is XORed with the plaintext (P) to produce the ciphertext (C). The decryption function takes the key and IV and produces the plaintext after XORing the ciphertext with the keystream.

Morpheus is key flexible as mentioned above and it consists of four words of 32 or 64 or 128 bits each that form an initial state. The state is broken up into groups: 4 “new” state words, which participate in the block update function, and 4 “old” state words, that are only used in the keystream output function. As shown in Figure 1a, a single round of Morpheus consists of adding (modulo 2) one new state word with a key material, adding (modulo 2^w) a second new state into the first and rotating the second word. The first word is an input to a lookup table, or substitution box (S-Box).

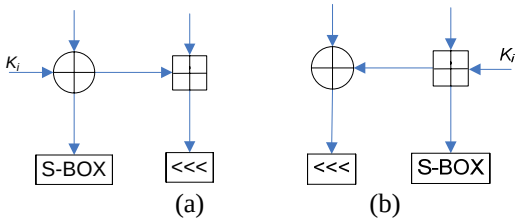

Figure 1: Single round on Morpheus

However, Morpheus changes its form in the round as illustrated in Figure 1b. Morpheus now consists of adding (modulo 2^w) one new state with a key material, adding (modulo 2) a second state into the first and rotating the second word. The first word is also an input to an S-Box. The new states are shown as vertical lines in Figure 1. Two rounds are applied in a diagonal pattern to the new state and create one block (see Figure 2).

Each block is represented by a unique number i. For the i-th block, the input state is denoted by $Z_0^{(i)}$, $Z_1^{(i)}$, $Z_2^{(i)}$, $Z_3^{(i)}$ and the output state by $Z_0^{(i+1)}$, $Z_1^{(i+1)}$, $Z_2^{(i+1)}$, $Z_3^{(i+1)}$, which forms the input to the next block with number i+1. During the block i, one word of keystream (S_i) is generated and four words of key material are added ($K_{i,0}$, $K_{i,1}$, $K_{i,2}$, and $K_{i,3}$).

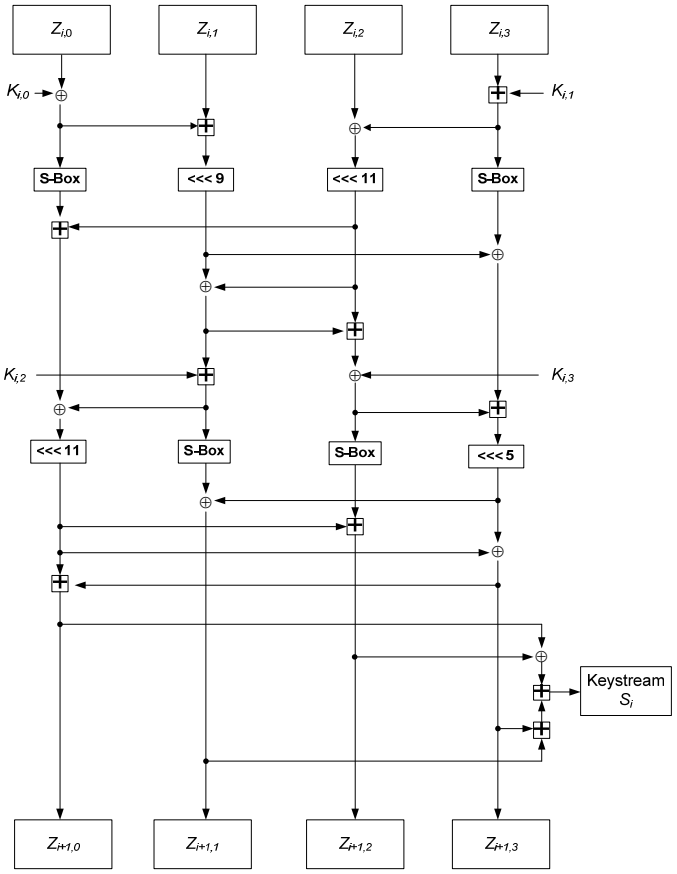

Figure 2: one block of Morpheus encryption

All states can be 32-bit or 64-bit or 128-bit words according to the key length and for the clarity of this paper we will consider 128-bit key that results to 32-bit words. In Morpheus, exclusive or is denoted by ⊕, addition modulo 2^w is denoted by +, left rotation is denoted by <<<, and lookup table is denoted by S-Box. The cipher block function F is defined as:

Function F ($Z_0^{(i)}$, $Z_1^{(i)}$, $Z_2^{(i)}$, $Z_3^{(i)}$, $K_{i,0}$, $K_{i,1}$, $K_{i,2}$, $K_{i,3}$)
Begin

$Z_0^{(i+1)} := Z_0^{(i)} \oplus Ki_{,0}$;	$Z_0^{(i+1)} :=$ S-Box($Z_0^{(i+1)}$);
$Z_3^{(i+1)} := Z_3^{(i)} + K_{i,1}$;	$Z_3^{(i+1)} :=$ S-Box($Z_3^{(i+1)}$);
$Z_1^{(i+1)} := Z_1^{(i)} + Z_0^{(i+1)}$;	$Z_1^{(i+1)} := Z_1^{(i+1)} <<< 9$;
$Z_2^{(i+1)} := Z_2^{(i)} \oplus Z_3^{(i+1)}$;	$Z_2^{(i+1)} := Z_2^{(i+1)} <<< 11$;
$Z_0^{(i+1)} := Z_0^{(i+1)} + Z_2^{(i+1)}$;	$Z_2^{(i+1)} := Z_2^{(i+1)} \oplus K_{i,3}$;

$Z_3^{(i+1)} := Z_3^{(i+1)} \oplus Z_1^{(i+1)}$; $Z_1^{(i+1)} := Z_1^{(i+1)} + K_{i,2}$;
$Z_1^{(i+1)} := Z_1^{(i+1)} \oplus Z_2^{(i+1)}$; $Z_3^{(i+1)} := Z_3^{(i+1)} + Z_2^{(i+1)}$;
$Z_2^{(i+1)} := Z_2^{(i+1)} + Z_1^{(i+1)}$; $Z_0^{(i+1)} := Z_0^{(i+1)} \oplus Z_1^{(i+1)}$;
$Z_2^{(i+1)} := \text{S-Box}(Z_2^{(i+1)})$; $Z_2^{(i+1)} := Z_2^{(i+1)} + Z_0^{(i+1)}$;
$Z_1^{(i+1)} := \text{S-Box}(Z_1^{(i+1)})$; $Z_1^{(i+1)} := Z_1^{(i+1)} \oplus X_3$;
$Z_3^{(i+1)} := Z_3^{(i+1)} <<< 5$; $Z_3^{(i+1)} := Z_3^{(i+1)} \oplus Z_0^{(i+1)}$;
$Z_0^{(i+1)} := Z_0^{(i+1)} <<< 11$; $Z_0^{(i+1)} := Z_0^{(i+1)} + Z_3^{(i+1)}$;
Return $(Z_0^{(i+1)}, Z_1^{(i+1)}, Z_2^{(i+1)}, Z_3^{(i+1)})$;
End.

Given the function F, one round of encryption is computed as follows:

$(Z_0^{(i+1)}, Z_1^{(i+1)}, Z_2^{(i+1)}, Z_3^{(i+1)}) := F(Z_0^{(i)}, Z_1^{(i)}, Z_2^{(i)}, Z_3^{(i)}, K_{i,0}, K_{i,1}, K_{i,2}, K_{i,3})$

Each round produces one word of keystream $S_i := [(Z_0^{(i+1)} \oplus Z_2^{(i+1)}) + (Z_1^{(i+1)} + Z_3^{(i+1)})]$. The ciphertext words are defined by $C_i := P_i \oplus S_i$.

A. Initialization & Encryption

Morpheus takes two parameters as input values; a secret key of 128-bits and a publicly known 128-bits IV. The IV value is considered as a four word input IV = (IV_3, IV_2, IV_1, IV_0) where IV_0 is the least significant word. Likewise, the key is considered as a four word input K = (k_3, k_2, k_l, k_0), where k_0 is the least significant word. States $Z_0^{(i)}$, $Z_1^{(i)}$, $Z_2^{(i)}$, $Z_3^{(i)}$ are initialized with K and IV according to function H.

Function $H(k_0, k_1, k_2, k_3, IV_0, IV_1, IV_2, IV_3)$
Begin
$Z_0 := k_0 \oplus IV_3$; $Z_1 := k_1 + IV_2$;
$Z_2 := k_2 \oplus IV_1$; $Z_3 := k_3 + IV_0$;
Return (Z_0, Z_1, Z_2, Z_3);
End.

The resulting 32-bit values Z_0, Z_1, Z_2 and Z_3 are used as an initial input to Morpheus block function F. F is executed for three times without producing any output bits. On the first time, the round- keys $K_{0,j}$ are equal to constant values (see section round key scheduling), but on the next two times the round-keys $K_{1,j}$ and $K_{2,j}$ are generated by the key scheduling function, described in the next section. The final four 32-bit words are used as the initial state of the encryption rounds.

After the initialization, the plaintext is encrypted. Each block generates one word of keystream, which is used to encrypt one word of plaintext. Decryption is almost identical to encryption. The keystream S_i generated after the first application of the F function in each block is used to decrypt the ciphertext, producing the plaintext word. The implementation must insure that any unused bytes of the final plaintext word are taken as zero for purposes of computing the block function, regardless of the value of the extra keystream bytes.

B. Round Key Scheduling

The output states $Z_0^{(i+1)}$, $Z_1^{(i+1)}$, $Z_2^{(i+1)}$, and $Z_3^{(i+1)}$ of the initialization phase are used to generate the round keys $K_{i,j}$ according to function S.

Function S $(Z_0^{(i+1)}, Z_1^{(i+1)}, Z_2^{(i+1)}, Z_3^{(i+1)}, K_{i,0}, K_{i,1}, K_{i,2}, K_{i,3})$
Begin
$K_{0,0} := 0xf35A$; $K_{0,1} := 0xB718$;
$K_{0,2} := 0xC59A$; $K_{0,3} := 0xE46D$;
For $i \geq 1$ and $0 \leq j \leq 3$ do
$K_{i,j} := \{[(Z_j^{(i+1)} <<< 3) \oplus ((i \bullet (j+1) <<< 10)] + K_{i-1,j}\}$;
Return $(K_{i,0}, K_{i,1}, K_{i,2}, K_{i,3})$;
End.

Initially, $K_{0,0}$, $K_{0,1}$, $K_{0,2}$ and $K_{0,3}$ are randomly set to constant values 0xf32A, 0xB718, 0xC59A, 0xE46D, respectively. For the next 3 blocks, each round-key is calculated by XORing, adding modulo 2^{32} with variable values and rotating with fixed numbers.

C. S-box Design

Morpheus defines a 16 × 16 matrix of byte values (Table 1) that contains a permutation of all possible 8-bit values (i.e. 2^8 possible combinations = 256 values). Each individual byte its input is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-Box to select a unique 8-bit output value. For example, the hexadecimal value {67} references row 6, column 7 of the S-Box, which contains the value {C7}. Accordingly, the value {67} is mapped into the value {C7}.

Table 1 – S-Box in Hexadecimal Values

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	F1	93	A9	0B	6C	C9	AA	19	32	EF	03	8A	92	B6	49	35
	1	4A	4F	6B	3F	85	E6	B1	45	4E	94	CF	E3	0F	06	AB	78
	2	F5	6F	5F	53	C2	B7	9C	67	59	13	72	5C	8B	26	BF	B4
	3	62	83	E2	C6	86	9F	76	7A	AE	91	44	69	EB	FC	27	F2
	4	D5	3D	73	CB	71	81	25	31	98	0E	AF	9B	11	65	8D	1A
	5	1B	41	28	2F	02	88	C8	08	3E	1E	84	AD	D1	E7	39	D0
	6	B8	DA	A6	10	3A	80	A2	C7	04	C3	BC	0C	30	3C	FF	7F
	7	87	96	4D	01	F7	A8	34	A3	DE	E5	A1	0A	D8	F3	48	16
	8	A4	BB	1C	D6	43	BD	FB	CD	9D	C1	F6	E0	7B	4C	3B	1D
	9	82	6E	C4	99	8F	24	29	8C	CC	FE	95	D7	A0	B2	50	14
	A	36	B3	1F	EA	D9	DF	55	66	58	38	47	4B	20	D4	05	5B
	B	DB	D2	23	33	57	F8	E9	ED	6D	68	89	54	A7	56	52	61
	C	21	CE	2B	90	74	F0	2A	5D	97	A5	EC	F9	75	63	E8	BE
	D	60	12	5A	D3	2D	6A	9A	64	9E	0D	46	07	F4	22	70	18
	E	79	09	B0	5E	7E	40	BA	DD	B5	7C	EE	15	7D	CA	8E	2C
	F	FA	42	E4	B9	51	77	AC	C0	00	17	E1	FD	C5	2E	DC	37

The S-Box is constructed as follows:

1. Initialize the S-Box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02}, …, {0F}; the second row contains {10}, {11}, etc.; and so on. Thus the value of the byte at row x, column y is {xy}.
2. Use the Secure Hash Algorithm (SHA-256) to create 256-bit values for each byte in row and column, i.e., {00}, {01}, {02}, … etc. The MSB of the hash value is stored in the corresponding xy byte. When two MSBs are the same the next byte is chosen. The procedure carries on from the MSB to the LSB.
3. Statistically correct the columns and rows of the S-Box (i.e. balance with the frequency and block test the number of 0s' and 1s' in each row and column)

IV. An Analysis of Morpheus

Compared to many ciphers, Morpheus is relatively easy to implement in both software and hardware.

A. Implementation

If 32-bit addition, exclusive-or, and rotation functions are available, all the functions are easily implemented in software. A single round takes only one clock cycle to compute on most current Pentium CPUs because the superscalar architecture can perform an addition and/or XOR simultaneously with a 32-bit rotation [8]. A block of Morpheus takes 8 cycles plus some overhead for the handling of the plaintext, keystream and ciphertext. The processing overhead of Morpheus is due to the initialization. The key scheduling only need to be done once for each key value and thus does not reserves any processing.

Morpheus is also fast in hardware. The rotations incur no gate delays, only wiring delays although they do not consume routing resources in chip layouts. The keystream is generated after 8 additions, 9 XORs, 4 fixed rotations and 4 memory accesses to the S-Box. There are several techniques for high-speed implementation but a conservative estimate of a low-cost ASIC layout is 2.5ns per 32-bit adder, 0.5 ns per XOR and 1.5 ns per access [7], which adds up to 30.5 ns/output. This translates to more than 150 MByte per second, or just under 1.5 Gbit per second. We roughly estimate that such a design would consume fewer than 25,000. Such a design would acquire about a 2 clock overhead per packet in order to process the initialization.

B. Security Issues

For flexibility, Morpheus allows several key sizes to be used as there are many situations in which larger than 128 bits of key material are available. The small set of elementary operations that Morpheus makes is efficient on a large number of software platforms. The absence of variable rotations and multiplications makes Morpheus small and efficient in hardware as well. The number of state words (4) used for output was chosen so that the total amount of unknown internal state is always at least 128 bits, even after the keystream output.

Morpheus use a lookup table (S-Box) to provide the necessary nonlinearity in conjunction with mixing of XORs with additions. Neither of these operations can be approximated well within the group of the other. In addition, the diffusion in Morpheus is fast and the attacker has very little control over the state it is not possible to limit the diffusion of differences. In those areas where dynamic attacks are possible, we use a sequence of 8 blocks instead of 3 to ensure thorough mixing of the state words with the round-keys.

The key initialization and round-key scheduling when combined is an unkeyed bijective function. The purpose is to spread the available entropy over all round key words. The round key scheduling ensures that all four key words depend on the key material. Using a bijective mixing function ensures that no two 128-bit input keys lead to the same working key values.

One of the dangers of a stream cipher is that the keystream will be re-used. To avoid this problem the sender must ensure that each (*K*ey, *IV*) pair must be unique for every encryption. A single sender must use a new and unique *IV* for each message. Multiple senders that want to use the same key must employ a scheme that divides the *IV* space into non-overlapping sets, in order to ensure that the same *IV* is never used twice. If two different messages are ever encrypted with the same (*K*ey, *IV*) pair, Morpheus loses most of its security properties.

Like any stream cipher, Morpheus is a cryptographically strong pseudorandom number generator (PRGN). For every input it produces a stream of pseudorandom data. Using the full block function, we ran statistical tests on many candidate rotation count sets to see how these values would affect the ability of the block function to diffuse changes and mix together separate information within the internal state.

C. Statistical Tests

Morpheus is a strong cryptographic PRGN since the tests below were successfully passed. Among our tests, we considered different rotation numbers and test the keystream and ciphertext for:

- the proportion of ones and zeros in the keystream;
- the proportion of ones and zeros in the binary derivate of the keystream;
- the difference between the proportion of ones up to and including the point and the proportion of ones after the point is noted;
- the number of repetitions and the χ^2 value of subblocks of the keystream;
- the run in the keystream of all ones (block) and all zeros (gap);
- the comparison between the actual sequence complexity and the sequence complexity threshold value for a bit stream of the same length;
- the minimum number of bits required to re-create the whole keystream using a linear feedback shift register;

- the probability of ciphertext block change whenever any bit of the plaintext block changes;

Most rotation counts did pretty well but our carefully selected rotation count sets were slightly better than random ones.
Morpheus was tested in both synchronous and self-synchronous encryption modes and passed statistics with frequent re-initializations of 160 and 256 bits of ciphetext.

D. Cryptanalysis

A successful attack is considered when an attacker can either predict a keystream bit he has not seen with a probability slightly higher than 50%. A number of attacking methods were considered in this section. We have not yet discovered any method of attacking Morpheus.

1) Static analysis

A static analysis takes the keystream and tries to reconstruct the state and key. Several properties make this type of attack difficult. Even if the whole state is known, any four consecutive keystream words are fully random. This is because each $K_{i,0}$ key value affects S_i in a bijective manner, so for any given state and any sequence of $K_{i,1}$ words, there is a bijective mapping from K_i to S_i. A similar argument applies when the block function is computed backwards. Any attempt to recover the key, even if the state is known at a single point, must span at least 4 blocks.

2) Period length

The Morpheus internal state is updated continuously by the round-keys $K_{i,j}$. The $K_{i,j}$ values depends on the "old" key words Z_j, the previous round-key $K_{i-1,j}$, the block number i, and the input key length. All 8 key words and all 4 *IV* words affect the state every block.

The initialization and key scheduling also ensures that different (*K*ey, *IV*) pairs produce different key sequences. To demonstrate this, we look at the sequence S_i of key words in the order they are used. Given just part of the sequence S_i, without the proper index values i, we cannot recover the key, *IV* and block number. This is due to the fact that the "old states" and the *IV* are different in the key scheduling and initialization phases. So long as the IV is not repeating, the keystream should have an arbitrarily long period, up to maximum packet size of 2^{64} bytes. The non-repeating *IV* word values prevent the state from ever falling into a cycle.

3) State collisions

To avoid internal collisions in less than 2^{128} words of chosen plaintext, Morpheus added 4 "old" state words, increasing the internal state significantly to 256 bits. Thus, the unknown state remains at 128 bits even after outputting the keystream word within a block, so no collisions are reasonably expected within the security bounds.

4) Weak keys

Morpheus makes constant use of the words of the working key. An all-zero working key it effectively omits a few operations from the block function, but we have not discovered any possible attack based on it. Once again statistical tests were performed to the keystream when *K* and *IV* are set to zeros or pattern data (i.e. 0s and 1s).

5) Chosen input differential attacks

One powerful mode of attack is for the attacker to make small changes in the input values and look at how the changes propagate through the cipher.

In Morpheus, this can be done only with the key or the *IV*. In each case, the block function is applied multiple times to the input. In Morpheus, all the places where such attacks are possible have several consecutive blocks without any output. A change to the *IV*, such as is considered in [5], was thoroughly mixed into the state by the time the first keystream word was generated. A search found no useful differentials for 3 blocks of Morpheus, nor useful higher-order differentials.

6) Algebraic attacks over GF(2)

Linearization and general algebraic techniques have been used to successfully analyze stream ciphers [1, 4]. The two round functions in Morpheus that combine integer addition, XOR, rotation and substitution yield to impractical algebraic attacks that require more than 2^{128} steps.

V. Conclusions

The design of Morpheus is based on various cipher design principles that have resulted from the intense research taking place on this area of cryptography. It combines the aspect of having a long chain of simple operations, rather than a small chain of complex ones for applications that work both in software and hardware. However, ciphers that are based on simple operations (i.e. addition, XORing and rotation) are susceptible to linear and algebraic attacks. Morpheus design incorporates lookup table and round keys to break such linearity.

Extensive statistical tests were performed to the keystream and to the ciphertext with special and weak keys. Known cryptanalytic methods were also applied and to our knowledge we have not yet discovered any workable method of attacking Morpheus.

References

[1] Armknecht, F. A linearization attack on the Bluetooth key stream generator. *Cryptology ePrint Archive Report* 2002/191, 2002. http://eprint.iacr.org/2002/191.

[2] Bernstein, D. J. Salsa20. eSTREAM, *ECRYPT Stream Cipher Project Report* 2005/025.

[3] Biryukov, A. A New 128-bit Key Stream Cipher: LEX, eSTREAM. *ECRYPT Stream Cipher Project Report* 2005/013. 2005

[4] Courtois, N. and Pieprzyk, J. Cryptanalysis of block ciphers with over defined systems of equations. *In Advances in Cryptology ASIACRYPT2002*, Yuliang Zheng (editor) volume 2501 of *Lecture Notes in Computer Science*, pages 267-287. Springer-Verlag, 2002.

[5] Daemen, J., Govaerts, R., and Vandewalle, J. Resynchronisation weaknesses in synchronous stream ciphers. *In Advances in Cryptology-EUROCRYPT '93*, Tor Helleseth (editor) volume 765 of *Lecture Notes in Computer Science*, pages 159-167. Springer-Verlag, 1993.

[6] Dj. Golic, J. Dj. Modes of operation of stream ciphers. *In Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000*, Douglas R. Stinson and Stafford Tavares (editors) volume 2012 of *Lecture Notes in Computer Science*, pages 233-247. Springer-Verlag, 2000.

[7] Whiting, D., Schneier, B., Lucks, S., and Muller F. Phelix: Fast encryption and authentication in a single cryptographic primitive. *eSTREAM, ECRYPT Stream Cipher Project Report* 2005/020. http://www.schneier.com/phelix.html.

[8] Lipmaa, H. and Moriai, S. Efficient algorithms for computing differential properties of addition. In Mitsuru Matsui, editor, *Fast Software Encryption2001*, *Lecture Notes in Computer Science*. Springer-Verlag, 2001. Available from http://www.tcs.hut.fi/helger/papers/lm01/.

[9] Muller, F. Differential Attacks against the Helix Stream Cipher. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE'04*, volume 3017 of *Lecture Notes in Computer Science*, pages 94-108. Springer-Verlag, 2004.

[10] Wu, H. and Preneel, B. Differential Attacks against Phelix. *eSTREAM, ECRYPT Stream Cipher Project Report* 2006/056. http://www.ecrypt.eu.org/stream/.

[11] Wu, H. and Preneel, B. Attacking the IV Setup of Stream Cipher LEX. *eSTREAM, ECRYPT Stream Cipher Project Report* 2005/059. http://www.ecrypt.eu.org/stream/