Differential Attacks against the Helix Stream Cipher

Frédéric Muller

DCSSI Crypto Lab
51 boulevard de la Tour-Maubourg, 75700 Paris - 07 SP, France
frederic.muller@m4x.org

Abstract. In this paper, we analyze the security of the stream cipher Helix, recently proposed at FSE'03. Helix is a high-speed asynchronous stream cipher, with a built-in MAC functionality. We analyze the differential properties of its keystream generator and describe two new attacks. The first attack requires 2^{88} basic operations and processes only 2^{12} words of chosen plaintext in order to recover the secret key for any length up to 256 bits. However, it assumes the attacker can force nonces to be used twice. Our second attack relies on weaker assumptions. It is a distinguishing attack that detects internal state collisions after 2^{114} words of chosen plaintext.

1 Introduction

A stream cipher is a secret key cryptosystem that transforms a short random secret key K into a long pseudo-random sequence also called keystream, which is XORed to the plaintext to produce the ciphertext. Although it is possible to obtain a similar primitive with a block cipher in a "pseudo-random number generator" mode (like OFB or CFB [6]), it is generally not considered to offer optimal speed performances. To respond efficiency considerations, fast stream ciphers reveal useful in real-life applications, especially those using live data transmission. Many recent stream ciphers proposals have been made in that direction including SEAL [16], SNOW [2], Scream [10] or Sober-t32 [11].

However, the security of stream ciphers is still an issue (see [1, 3, 7]), especially when compared to the level of confidence in block ciphers security. For instance, all stream ciphers candidates for the NESSIE project [14] revealed various degrees of weakness allowing at least distinguishing attacks faster than exhaustive search, while no second round block cipher was successfully attacked. As a consequence, NESSIE did not select any stream cipher in its final portfolio. Thus the actual challenge is to design fast stream ciphers and provide a better confidence in their security level. Several new ciphers aim at reaching these expectations.

Helix was recently proposed at FSE'03 [5]. It is an asynchronous stream cipher based on a fast keystream generator. Its advantage over other new ciphers is to offer both confidentiality and integrity. Indeed, after encryption, Helix can

B. Roy and W. Meier (Eds.): FSE 2004, LNCS 3017, pp. 94–108, 2004.
© International Association for Cryptologic Research 2004

produce a tag that guarantees the integrity of the message for very little additional computation and without requiring a second pass. This functionality is very useful in many applications where encryption and authentication must function together on streaming data. Recently, several block cipher modes of operation also providing integrity "almost for free" (see [9, 12, 15]) have been proposed, but some of them appear to be patented, which is supposedly not the case of Helix.

Moreover, the analysis of Helix is an interesting topic since new mechanisms that will be included in the new 802.11i standard for wireless networks are apparently fairly close to Helix [4, 18]. The new standard will have to repair some cryptologic flaws from the previous 802.11b standard, which resulted from weaknesses in RC4 key scheduling and from an improper use of initialization vectors [8].

In this paper, we analyze the security of Helix against chosen plaintext and chosen nonce attacks. We present two attacks which are both faster than exhaustive search. Our first attack recovers the secret key (for any length up to 256 bits) with time complexity of 2^{88} basic operations and using 2^{12} words of chosen plaintext. It assumes an attacker could force encryption of several messages using the same pair (key,nonce). Our second attack is based on internal state collisions and distinguishes Helix from random with data complexity of 2^{114} blocks. This attack uses chosen nonces and chosen plaintext but never re-uses a pair (key,nonce). Our paper is organized as follows : first, we briefly describe Helix. Then, in Section 3, we show two weaknesses of the cipher which are further developed in Section 4. In Section 5, we describe two attacks based on the previous observations.

2 Description of Helix

Helix offers two main features : encryption of a plain message and production of a Message Authentication Code (MAC) to ensure integrity. Several modes of operation for Helix are proposed by its authors - encryption only, MAC only, PRNG, ... Here, we describe briefly the mechanisms of Helix that are important in our attacks. More details about this design can be obtained in [5].

We mostly handle 32 bits values that we denote as *words*. Besides, $\oplus$ denotes bitwise addition on these values and $+$ addition modulo 2^{32}. $ROTL_n(x)$ is the circular rotation of the word x by n bits to the left. We also use the notations LSB and MSB to refer to the least and most significant bit of a word.

2.1 General Structure of the Cipher

Helix is an asynchronous stream cipher, based on an iterated block function applied to an internal state of 160 bits. The input consists in a secret key K of varying length, up to 256 bits, and a nonce N of 128 bits. The internal state before encryption of the i-th word of plaintext is represented as 5 words

$$(Z_0^{(i)}, \ldots, Z_4^{(i)})$$

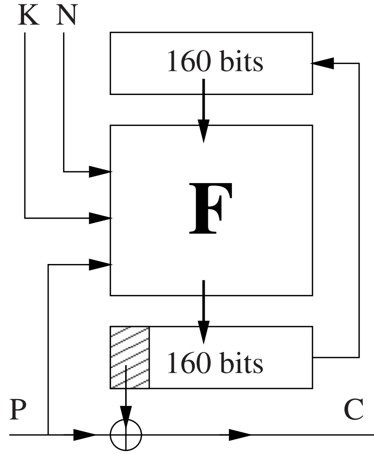

Fig. 1. The general structure of Helix

which are initialized for $i = 0$ using K and N. Details of this initialization mechanism are irrelevant here. The general structure of the encryption algorithm is described in Figure 1. It basically uses a block function F to update the internal state in function of the plaintext P, the key K and the nonce N.

More precisely, during the i-th round, the internal state is updated with F, using the i-th word of plaintext P_i and two words derived from K, N and i, denoted as $X_{i,0}$ and $X_{i,1}$. We refer to them as the "round key words". Hence,

$$(Z_0^{(i+1)}, \ldots, Z_4^{(i+1)}) = F(Z_0^{(i)}, \ldots, Z_4^{(i)}, P_i, X_{i,0}, X_{i,1})$$

The i-th keystream word, also denoted as S_i, is equal to $Z_0^{(i)}$. It is added to P_i to produce the i-th ciphertext word C_i. Thus,

$$S_i = Z_0^{(i)}$$
$$C_i = S_i \oplus P_i$$

This process is repeated until all words of the plaintext have been encrypted. Finally, a last step (described in [5]) can generate a tag of 128 bits that constitutes the MAC. More details on this general framework are given in the following sections.

2.2 The Block Function

The round function F of Helix mixes three types of basic operations on words: bitwise addition represented as $\oplus$, addition modulo 2^{32} represented as $\boxplus$, and cyclic shifts represented as $<<<$. F relies on two consecutive applications of

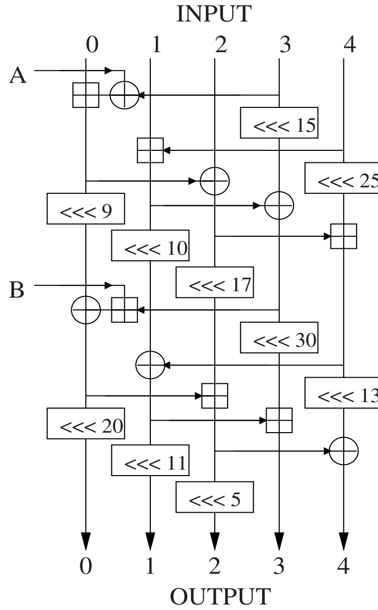

Fig. 2. The half-round "helix" function G

a single "helix" function, which constitutes half of the round function. This "helix" function is denoted as G and is represented in Figure 2.

G uses two auxiliary inputs (A, B). In the first half of the round function, $(A, B) = (0, X_{i,0})$ and in the second half, $(A, B) = (P_i, X_{i,1})$. Thus, the block function can be described by the following relations

$$(Y_0^{(i)}, \ldots, Y_4^{(i)}) = G(Z_0^{(i)}, \ldots, Z_4^{(i)}, 0, X_{i,0})$$
$$(Z_0^{(i+1)}, \ldots, Z_4^{(i+1)}) = G(Y_0^{(i)}, \ldots, Y_4^{(i)}, P_i, X_{i,1})$$

where $(Y_0^{(i)}, \ldots, Y_4^{(i)})$ is the internal state in the middle of the computation.

2.3 Role of *K* and *N*

To protect the cipher against related-key attacks, a first step is applied that computes a working key K from the actual secret key U. Independently of its length $l(U)$, K is always 256 bits long and is used in all subsequent operations instead of U. The derivation of K is based on 8 rounds of a Feistel network. The result is also represented as 8 words: $K_0, \ldots, K_7$.

Besides, Helix uses a nonce N to obtain different keystream sequences with the same secret key. N is always 128 bits long and is generally represented as 4 words: $N_0, \ldots, N_3$. An expansion phase turns it into a 256 bits value by creating 4

additional words $N_4, \dots, N_7$ defined as

$$N_{k+4} := (k \bmod 4) - N_k$$

for $k = 0, \dots, 3$. During the i-th round of encryption, the round key words $X_{i,0}$ and $X_{i,1}$ are computed as

$$\begin{aligned} X_{i,0} &:= K_{i \bmod 8} \\ X_{i,1} &:= K_{(i+4) \bmod 8} + N_{i \bmod 8} + X'_i + i + 8 \\ X'_i &:= \begin{cases} \lfloor (i+8)/2^{31} \rfloor & \texttt{if } i \bmod 4 = 3 \\ 4\, l(U) & \texttt{if } i \bmod 4 = 1 \\ 0 & \texttt{otherwise} \end{cases} \end{aligned}$$

These values depend only on i, K and N_i. Besides, it is straightforward to reconstruct the secret key from these values for 4 consecutive rounds when the nonce is known.

3 Some Weaknesses of Helix

In this section, we describe two weaknesses of the block function. They respectively concern the role of the plaintext words and the nonce words at each round.

3.1 Influence of Each Plaintext Word

Since Helix requires a plaintext-dependent keystream, it is reasonable to analyze the round function assuming an attacker can control the plaintext introduced. In general, an attacker should not be able to recover any information about the secret key or the internal state of the cipher, by observing the keystream corresponding to chosen plaintext.

Using the notations of Section 2, P_i denotes the i-th word of plaintext. It is introduced inside Helix internal state at the i-th advance. Then, at the beginning of the $(i+1)$-th advance, a new keystream word S_{i+1} is produced. From the description of Helix, one sees that P_i is introduced only in the second half of the block function (as the input A of Figure 2). It is XORed to $Y_3^{(i)}$, then added to $Y_0^{(i)}$. The result is then modified only once before the end of the round - excepting cyclic shifts - through a XOR with some intermediate value (referred to as a). However, it is easy to verify that a is actually independent of P_i. Thus S_{i+1} can be computed as

$$S_{i+1} = Z_0^{(i+1)} = ROTL_{20}(a \oplus ROTL_9(Y_0^{(i)} + (Y_3^{(i)} \oplus P_i)))$$

If the plaintext word $P'_i = P_i \oplus \Delta$ was introduced instead of P_i, then the next keystream word would be S'_{i+1}, such that

$$\begin{aligned} \delta &= S_{i+1} \oplus S'_{i+1} \\ &= ROTL_{29}((x + (y \oplus P_i)) \oplus (x + (y \oplus P_i \oplus \Delta))) \end{aligned}$$

where x and y respectively denote the intermediate words $Y_0^{(i)}$ and $Y_3^{(i)}$. Suppose that $P_i = 0$, then for any difference Δ on the plaintext,

$$\Delta' = ROTL_3(\delta) = (x + y) \oplus (x + (y \oplus \Delta)) \quad (1)$$

is the corresponding difference on the keystream. In Section 4, we will discuss how an attacker can take advantage of this differential property.

3.2 Influence of Each Nonce Word

Similar differential properties hold regarding each nonce word. Indeed, the nonce N serves two purposes in Helix :

– Fill the initial 160 bits of internal state.
– Derive two words $X_{i,0}$ and $X_{i,1}$ introduced at round i.

Concerning this second task, it appears from Section 2.3 that the two "key words" introduced at each round do not depend on the full nonce. Actually, the round key words at round i depend only on $N_{i \bmod 4}$. Therefore, if we consider two distinct nonces N and N' where only one word changes, the round function will essentially apply the same mapping on the internal state, for 3 rounds out of 4. This property has consequences on the propagation of state collisions.

Moreover, if only one nonce word N_i is modified to $N_i + \Delta$ then, for rounds j such that $j \bmod 4 \neq i$, both round key words remain unchanged. For other positions, $X_{j,1}$ is changed to $(X_{j,1} \pm \Delta)$ while $X_{j,0}$ is unchanged. Since $X_{j,1}$ is introduced at the very end of the block function, we have a differential property, like in Section 3.1. When all other inputs are unchanged, the difference on the keystream words resulting from this difference Δ on the nonce word N_i is

$$\Delta' = a \ \oplus (a \pm \Delta) \quad (2)$$

for some unknown internal value a (see Figure 2).

4 Differential Properties of Addition Modulo 2^{32}

We have seen that differential patterns on the plaintext or the nonce propagate to simple differential patterns on the keystream. More precisely, the differential property on the plaintext is related to a general problem concerning linear approximations of addition modulo 2^{32} that can be summarized by relation (1). In this section, we will describe various ways to take advantage of this observation.

4.1 Related Problems

A well known problem (see [13]) is, given two fixed words x and y, to find a pair (Δ, Δ') such that

$$\Delta' = (x + y) \oplus (x + (y \oplus \Delta)) \quad (3)$$

and that is observed with high probability. This problem has been studied from a theoretical point of view in [17]. However, in the present situation, we are looking things the other way around since x and y are unknown to us but we might be able to choose Δ and observe Δ'. More precisely, we want to

1. find statistical properties that can be easily detected in order to distinguish Helix from a random source.
2. recover some secret information about the internal state of Helix (the values of x and y for instance).

4.2 A "Dummy" Distinguisher

Suppose an attacker encrypts two messages that begin similarly, but, at some point, differ on one word by

$$\Delta = \texttt{0x80000000}$$

Then, the difference on the next keystream word (called Δ') is such that $\Delta' = \Delta$, since there is no propagation from MSBs to LSBs during an addition. Using this relation, the block function of Helix can be distinguished from a random source with two chosen messages, but this requires to use twice the same key and the same nonce. This attack scenario is discussed in Section 5. In the next section, we go further by trying to actually recover the two internal values x and y using relation (3).

4.3 Recovering x and y

In this section, we are interested in recovering the two intermediate values x and y involved in relation (3). Thus, we have to consider the following problem

> *Problem 1.* Let x and y be two given constants of 32 bits. For any Δ,
>
> $$\Delta' = (x + y) \oplus (x + (y \oplus \Delta)) \tag{4}$$
>
> is given. How many (x, y) are possible solutions ? Give an efficient algorithm to recover these solutions.

First, it is easy to see that the solution is not always unique. Indeed, if $x = 0$, then Δ' does not depend on y. However, in average, the number of candidates is small. In this section, we propose an efficient algorithm to recover the two unknown values x and y with a limited number of observations. The following notations are used : w_j denotes the j-th bit of a word w. Besides, let c_j denote the carry bit at position j in the addition of x and $y \oplus \Delta$. For all j, $0 \leq j \leq 31$,

$$(x + (y \oplus \Delta))_j = x_j \oplus y_j \oplus \Delta_j \oplus c_j$$

and initially $c_0 = 0$. We also suppose that $x \neq 0$.

Claim. Let t, $0 \leq t \leq 30$, denote the position of the least significant bit '1' of x. Then, there are exactly 2^{t+3} valid pairs (x, y), solutions of the previous problem. Recovering these solutions can be done by testing at most 93 chosen values of Δ.

We use the following induction

- Assume all bits of x and y are known up to position $(i - 1)$.
- If any $x_j = 1$ with $0 \leq j < i$, then
  - By choosing an appropriate value of Δ_k for $j \leq k < i$, it is possible to obtain any value of c_i (0 or 1), since everything is known up to position i.
  - In both cases, pick both values of Δ_i (0 and 1) and set all other bits of Δ to 0. The resulting value of Δ'_{i+1} depends only on the carry bit c_{i+1}.
  - Recover x_i and y_i by comparing the different distributions (see Table 1)
- Otherwise
  - Necessarily, $c_i = 0$
  - Using Table 1, it is still possible to recover x_i.
  - No information on y_i is obtained.

Therefore, by induction, all bits of x can be recovered from position 0 to 30 (it is impossible to recover x_{31} because no observation can be made about position 32 of Δ'). Similarly, all bits of y from position $(t + 1)$ to 30 can be recovered. The other $t + 3$ bits of x and y need to be guessed. When $x = 0$, our analysis remains valid by taking $t = 30$.

In fact, 3 queries are enough to distinguish the distributions in Table 1. Thus, at most $3 \times 31 = 93$ queries are sufficient to recover a valid solution (x, y). Besides, it is easy to verify that flipping the bit y_t will imply to flip all bits x_j and y_j for $t < j < 31$ in order to obtain an other valid solution, since all carry bits also get flipped. Therefore all solutions of the system can be expressed directly from a single solution, without any extra query.

We performed some experiments using various values of x and y and always identified with success the expected number of 2^{t+3} solutions.

Table 1. Distribution of Δ'_{i+1} depending on x_i and y_i

x_i	y_i	c_i	Δ_i	Δ'_{i+1}
1	1	0	0	δ
1	1	0	1	$\delta \oplus 1$
1	1	1	0	δ
1	1	1	1	δ
1	0	0	0	δ
1	0	0	1	$\delta \oplus 1$
1	0	1	0	$\delta \oplus 1$
1	0	1	1	$\delta \oplus 1$

x_i	y_i	c_i	Δ_i	Δ'_{i+1}
0	1	0	0	δ
0	1	0	1	δ
0	1	1	0	$\delta \oplus 1$
0	1	1	1	δ
0	0	0	0	δ
0	0	0	1	δ
0	0	1	0	δ
0	0	1	1	$\delta \oplus 1$

5 Attacks against Helix

In this section, two attacks against Helix are developed. The first one is a distinguishing attack using chosen plaintext, which is extended to a key recovery attack requiring 2^{88} basic operations and about 2^{12} block encryptions. A second attack takes advantage of choosing similar nonces to detect internal state collisions.

5.1 A Distinguishing Attack

In Section 3.1, we have shown that the introduction of a chosen difference on the plaintext from a fixed internal state results in predictable patterns on the keystream. However, to turn these observations into an attack, it is necessary to consider the following scenario

- The attacker requests encryption of some random message $P = (P_1, \ldots, P_n)$ under some pair (key,nonce) $= (K, N)$. The resulting ciphertext is $C = (C_1, \ldots, C_n)$.
- He requests encryption with (K, N) of an other message where P_{n-1} is replaced by $P'_{n-1} = P_{n-1} \oplus \Delta$. This yields the ciphertext $C' = (C'_1, \ldots, C'_n)$.
- The attacker observes $\Delta' = C_n \oplus C'_n$.

In this case, we have seen that a real Helix output can be distinguished from a random output, by picking $\Delta =$ `0x80000000` (then, necessarily, $\Delta' = \Delta$).

5.2 A Simple Key Recovery Attack

Now, we wish to extend the observations of Section 4.3. This technique allowed an attacker to retrieve up to 64 bits of intermediate values by observing the keystream corresponding to well chosen plaintexts. Actually, this information leakage is an important weakness, since it reduces the entropy of the internal state. Using an appropriate guessing technique, one may hope to turn it into a key recovery attack. Such an attack is generally called a *guess-then-determine attack*, since an attacker will first *guess* some internal state bits and then *determine* the correct guess using available information.

First, let us consider the round number i of Helix encryption. We suppose an attacker has access to the keystream word $Z_0^{(i)}$ and to a few candidates for $Y_0^{(i)}$ and $Y_3^{(i)}$ as described in Section 4.3. These two intermediate words depend on the internal state at input of round i : $(Z_0^{(i)}, \ldots, Z_4^{(i)})$ and on the first round key word $X_{i,0}$. This is represented in Figure 3 where each box is a 32 bits value and dashed boxes represent known values. An attacker may hope to use these conditions to reduce the number of possible internal states to

$$2^{128} \times 2^{32} \times 2^{-64} = 2^{96}$$

Actually, this number can be reached by guessing $Z_2^{(i)}$, $Z_3^{(i)}$ and $X_{i,0}$. Then the attacker can retrieve $Z_1^{(i)}$ and $Z_4^{(i)}$ by looking precisely at the function G

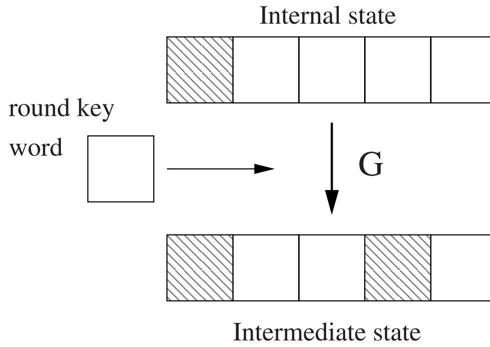

Fig. 3. The framework of the simple attack

(see Figure 2). Thus, the attacker can indeed find 2^{96} candidates for the internal state at the beginning of round i. To tell which candidate is correct, some of the previous rounds (say $\tau = 5$ rounds) need to be inverted. This can be done without increasing the number of candidates, provided $Y_0^{(i-j)}$ and $Y_3^{(i-j)}$ are known, for $0 \leq j < \tau$. For this purpose, the recovery technique of Section 4.3 needs to be applied τ times here. As long as it returns few solutions, an appropriate round inversion reduces the number of candidates - roughly by a factor 2^{32}. Thus, for $\tau = 5$ we eventually obtain a unique candidate, and enough "round key words" to directly retrieve the complete secret key.

To summarize, this simple attack requires to guess 96 bits of internal state and to apply τ times the technique described in Section 4.3 to recover intermediate values. However, this technique does not provide a unique solution, which increases the time complexity of the attack. Actually, only the round i is in the critical path and with probability $\frac{1}{2}$, the number of solutions here is only 8. In this "good" case, the complexity of the attack is $2^{96} \times 8 = 2^{99}$ basic instructions. In "bad" cases, there are more than 8 solutions at position i, but the attacker may easily find another position i' where there are only 8 solutions.

The data complexity corresponds to the encryption of $\tau \times 93$ pairs of messages of length at most $\tau = 5$ words. Thus, the number of plaintext blocks encrypted is

$$2 \times 5 \times 5 \times 93 \simeq 2^{12}$$

5.3 An Improved Attack

A more subtle guessing technique can be applied using bitwise analysis of the block function. The "subtle" attack consists in guessing only 2 words, $Z_3^{(i)}$ and $(Z_1^{(i)} + Z_4^{(i)})$, plus 17 LSBs of $Z_2^{(i)}$. Then, like in the "simple" attack, the attacker can obtain the 17 LSBs of $ROTL_{25}(Z_4^{(i)})$ and thus the 17 LSBs of $ROTL_{25}(Z_1^{(i)})$. Looking at the block function of round $i-1$, the attacker knows two output

words, and has partial knowledge of the three other output words. Two relations can be written, involving one unknown intermediate word a

$$\begin{aligned} Z_3^{(i)} &= ROTL_{21}(Z_1^{(i)}) + a \\ Y_3^{(i-1)} &= ROTL_{28}(Z_1^{(i)}) \oplus ROTL_{21}(Z_2^{(i)}) \\ &\quad \oplus ROTL_{26}(Z_4^{(i)}) \oplus ROTL_{19}(a) \end{aligned}$$

From the first relation, one sees that guessing the 4 LSBs of a will give the attacker a candidate for the 21 LSBs of a (using partial knowledge of $Z_1^{(i)}$). Then, using the second relation, a condition on bit number 13 of $Y_3^{(i-1)}$ is obtained. This condition eliminates half of the candidates. Then, each additional guessed bit of $Z_2^{(i)}$ provides one extra condition, that is immediately used to discard half of the guesses. This "early abort" technique results in a guessing complexity of

$$2^{32} \times 2^{32} \times 2^{17} \times 2^4 = 2^{85}$$

The backtracking can be performed here exactly as before to complete the attack. The resulting time complexity is reduced to $8 \times 2^{85} = 2^{88}$ guesses (each requiring a few boolean operations on 32 bit words). Furthermore, the existence of even better guessing techniques should be investigated.

5.4 Practical Impact

Previously, we have proposed a differential attack on Helix, using chosen plaintext. It requires to obtain twice the same internal state as input of the block function. Thus, the attacker needs to encrypt twice with the same key and the same nonce, and to introduce a difference in the plaintext at some point. However in [5], it is specified that "the sender must ensure that each (K, N) is used at most once to encrypt a message", otherwise Helix "loses its security properties". According to the authors, this requirement is not restrictive since it is underlying many similar situations in cryptography. For instance, when using a synchronous stream cipher, if secret key and nonce are unchanged, the same pseudo-random sequence is generated twice, which breaks the confidentiality. Similar problems may also be encountered when using a block cipher in OFB mode for instance. In general, a distinguishing attack is always possible when nonces are re-used. We believe the situation is more preoccupying in the case of Helix since we obtain key recovery attacks and not only distinguishing attacks.

On the one hand, there are situations where the previous scenario is not realistic. Indeed, the secret key may be used to communicate only in one direction. In this case, it is straightforward for the sender never to re-use the same nonce (he can use counters for instance). Apparently, this is true for wireless networks, where each pair of users have two separate secret keys, one for each direction. A differential attack cannot be applied there, unless the attacker gains physically access to the encryption machine and can force nonce repetition. This may be possible in some particular occasions, but in general it is a strong assumption.

On the other hand, in most situations, our differential attack scenario seems realistic. For instance, several users often need to share a secret key. Even if they split properly the nonce space, what happens if the same message is sent to multiple receivers ? An attacker can sit in the middle, and modify the ciphertext on one of the communication channels. Then, by comparing a “faulty” decryption with a correct decryption, he may obtain the kind of differential information he needs.

To conclude, we think the security impact of our attacks will highly depend on the context, but in general, one should expect the block function of Helix to resist better against differential attacks. Overall, the secrecy of the key cannot reasonably rely on the absence of nonce repetition.

5.5 A Chosen Nonce Attack

A weakness regarding the influence of each nonce word has been identified in Section 3.2. Here, we propose an extension to a distinguishing attack against Helix. Its complexity is much bigger than the previous attack. However it has the advantage of being based on weaker assumptions. Indeed, in this case, the attacker does not need to encrypt several messages with the same pair (key,nonce). Instead, we suppose that the same plaintext P is encrypted twice with the same secret key, but two distinct nonces N and N' such that

$$N = (N_0, N_1, N_2, N_3)$$
$$N' = (N_0, N_1, N_2, N_3 + \Delta)$$

Then, as argued in 3.2, the block function is essentially the same for any round i such that $i \bmod 4 \neq 3$. If a state collision occurs on the input of such a round, it will also propagate to a state collision for the input of the next round. Thus state collisions on inputs of rounds i such that $i \bmod 4 = 0$ imply collisions on 4 consecutive blocks of keystream. Moreover, the difference on the 5-th block can be predicted exactly (by picking $\Delta = 10\ldots0_x$ for instance). Thus, we obtain a detectable condition on 160 bits of keystream. This is sufficient to detect state collisions with good probability.

Therefore, contrarily to what is claimed in [5], state collisions in Helix can be detected. However, the length of messages is not allowed to exceed 2^{62} blocks, so collisions are unlikely to be observed for practice purpose.

5.6 Forcing the Collisions

In this section, we show that the previous attack can be extended into a distinguisher against Helix with only 2^{114} encrypted blocks. This is an important result, since it constitutes a break of the cipher, according to the definition given by the authors [5].

The general idea is to work on a large set of nonces that will preserve collisions during a few rounds. Then these collisions can be detected by observing the corresponding keystream blocks. More precisely, we build a message P of the

maximal authorized length 2^{62} words by repeating 2^{62} times the same word P_0. Then, P is encrypted under a fixed unknown secret key K using different nonces of the form

$$N^{(\delta,\Delta)} = (N_0 + \delta, N_1 + \delta, N_2 + \delta, N_3 + \Delta)$$

with four fixed constants $(N_0, \ldots, N_3)$. δ is of the form $8 \times x$ where x spans all values from 0 to 2^{20} and Δ spans all 2^{32} possible words. Therefore the number of blocks encrypted is

$$2^{62} \times 2^{32} \times 2^{20} = 2^{114}$$

As before, we consider any state collisions that occurs between two different nonces $N^{(\delta_1,\Delta_1)}$ and $N^{(\delta_2,\Delta_2)}$, at two different positions in the encryption, respectively i_1 and i_2. We would like this state collision to be preserved for several rounds, in order to detect some properties on the keystream, as in the previous Section. We are sure that the plaintext word introduced is always P_0, by construction. Furthermore we would like to have the same round key words for both encryptions. Hence, these positions should satisfy

$$i_1 \bmod 8 = i_2 \bmod 8 = 0$$

in order to have $X_{i_1+j,0} = X_{i_2+j,0}$ for all j. Besides, if

$$\delta_1 + i_1 = \delta_2 + i_2 \bmod 2^{32} \tag{5}$$

then $X_{i_1+j,1} = X_{i_2+j,1}$ when $j \bmod 4 \neq 3$. In this case, the state collision is preserved during at least 3 rounds. Concerning rounds $i_1 + 3$ and $i_2 + 3$, we would like to also preserve the collision, thus we need $X_{i_1+3,1} = X_{i_2+3,1}$ or

$$\Delta_1 + i_1 + X'_{i_1+3} = \Delta_2 + i_2 + X'_{i_2+3} \bmod 2^{32} \tag{6}$$

With these three assumptions, the state collision is preserved at least until the rounds $i_1 + 7$ and $i_2 + 7$ which results in collisions on 8 consecutive words of keystream.

To mount an attack, we first store sequences of 8 consecutive keystream words, for each message and for each position i such that $i \bmod 8 = 0$. Then, we look for a collision among the $\frac{2^{114}}{8} = 2^{111}$ entries in this table. This can be achieved by sorting the table, with complexity of $2^{111} \times 111 \simeq 2^{118}$ basic instructions. Then, since we consider objects of 256 bits, the number of "fortuitous" collisions in the table is

$$\frac{2^{111} \times 2^{111}}{2} \times 2^{-256} \simeq 0$$

Besides, when a "true" state collision occurs, a collision is also observed on the entries of the table, provided the additional assumptions (5) and (6) hold. (5) holds with probability 2^{-29}, since all terms are multiples of 8, and (6) holds with probability 2^{-32}. Therefore, the number of "true" collision observed in the table is in average

$$\frac{2^{111} \times 2^{111}}{2} \times 2^{-160} \times 2^{-29} \times 2^{-32} = 1$$

Thus we have considered enough encrypted data to detect some particular state collisions that are preserved during a few rounds. We achieve it by observing patterns of 8 consecutive words of keystream. For a true Helix output we expect to find a collision in the previous table, while it will not be the case for a random output. Actually this distinguishing attack can be slightly improved if we take into account the case $i \bmod 8 = 4$.

To conclude, we have proposed a distinguishing attack against Helix requiring the encryption of 2^{114} words of plaintext under chosen nonces. This attack is faster than exhaustive search, processes less than 2^{128} blocks of plaintext and respects the security requirements proposed in [5], since no pair (key,nonce) is ever re-used to encrypt different messages. Therefore, this attack constitutes a theoretical break of Helix.

6 Conclusion

This paper describes two attacks against the new stream cipher Helix. The first one recovers the secret key with a reasonably low complexity in time and data, so we think it should be considered as an important threat. The assumptions we use are quite usual (chosen plaintext, chosen nonce), but they are outside the security model proposed by the authors of the cipher.

However, we also propose a second attack, less efficient but which relies on weaker assumptions. This distinguishing attack constitutes a break of Helix according to the definition given by the authors. Both attacks result from weak differential properties of the encryption function regarding the plaintext and the nonce. In general, our attack illustrates the fact that one should be careful to protect new stream ciphers against differential-like attacks.

References

[1] D. Coppersmith, S. Halevi, and C. Jutla. Cryptanalysis of Stream Ciphers with Linear Masking. In M. Yung, editor, *Advances in Cryptology – Crypto'02*, volume 2442 of *Lectures Notes in Computer Science*, pages 515–532. Springer, 2002. 94

[2] P. Ekdahl and T. Johansson. SNOW - a New Stream Cipher. In *First Open NESSIE Workshop, KU-Leuven*, 2000. Submission to NESSIE. Available at `http://www.it.lth.se/cryptology/snow/`. 94

[3] P. Ekdahl and T. Johansson. Distinguishing Attacks on SOBER-t16 and t32. In J. Daemen and V. Rijmen, editors, *Fast Software Encryption – 2002*, volume 2365 of *Lectures Notes in Computer Science*, pages 210–224. Springer, 2002. 94

[4] N. Ferguson. Michael: an improved MIC for 802.11 WEP. Document 2-020. Available at `http://grouper.ieee.org/groups/802/11/`. 95

[5] N. Ferguson, D. Whiting, B. Schneier, J. Kelsey, S. Lucks, and T. Kohno. Helix, Fast Encryption and Authentication in a Single Cryptographic Primitive. In T. Johansson, editor, *Fast Software Encryption – 2003*, 2003. To appear. 94, 95, 96, 104, 105, 107

[6] FIPS PUB 81. *DES Modes of Operation*, 1980. 94

[7] S. Fluhrer. Cryptanalysis of the SEAL 3.0 Pseudorandom Function Family. In M. Matsui, editor, *Fast Software Encryption – 2001*, volume 2355 of *Lectures Notes in Computer Science*, pages 135–143. Springer, 2001. 94
[8] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In S. Vaudenay and A. M. Youssef, editors, *Selected Areas in Cryptography – 2001*, volume 2259 of *Lectures Notes in Computer Science*, pages 1–24. Springer, 2001. 95
[9] V. D. Gligor and P. Donescu. Fast Encryption and Authentication: XCBC Encryption and XECB Authentication Modes. In M. Matsui, editor, *Fast Software Encryption – 2001*, volume 2355 of *Lectures Notes in Computer Science*, pages 192–108. Springer, 2001. 95
[10] S. Halevi, D. Coppersmith, and C. Jutla. Scream : a Software-efficient Stream Cipher. In L. Knudsen, editor, *Fast Software Encryption – 2002*, volume 2332 of *Lectures Notes in Computer Science*, pages 195–209. Springer, 2002. 94
[11] P. Hawkes and G. Rose. Primitive Specification and Supporting Documentation for SOBER-t32. In *First Open NESSIE Workshop*, 2000. Submission to NESSIE. 94
[12] C. Jutla. Encryption Modes with Almost Free Message Integrity. In B. Pfitzmann, editor, *Advances in Cryptology – Eurocrypt'01*, volume 2045 of *Lectures Notes in Computer Science*, pages 529–544. Springer, 2001. 95
[13] H. Lipmaa and S. Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In M. Matsui, editor, *Fast Software Encryption – 2001*, volume 2355 of *Lectures Notes in Computer Science*, pages 336–350. Springer, 2001. 99
[14] NESSIE - New European Schemes for Signature, Integrity and Encryption. `http://www.cryptonessie.org`. 94
[15] P. Rogaway, M. Bellare, J. Black, and T. Krovetz. OCB/ A Block-cipher Mode of Operation for Efficient Authenticated Encryption. In *Eight ACM Conference on Computer and Communications Security (CCS-8)*, pages 196–205. ACM Press, 2001. 95
[16] P. Rogaway and D. Coppersmith. A Software-optimized Encryption Algorithm. In R. Anderson, editor, *Fast Software Encryption – 1994*, volume 809 of *Lectures Notes in Computer Science*, pages 56–63. Springer-Verlag, 1994. 94
[17] J. Wallen. Linear Approximations of Addition Modulo 2^n. In T. Johansson, editor, *Fast Software Encryption – 2003*, 2003. To appear. 100
[18] IEEE P802.11, The Working Group for Wireless LANs. `http://grouper.ieee.org/groups/802/11/`. 95