Helix

Fast Encryption and Authentication in a Single Cryptographic Primitive

Niels Ferguson[1], Doug Whiting[2], Bruce Schneier[3], John Kelsey[4], Stefan Lucks[5], and Tadayoshi Kohno[6]

[1] MacFergus, `niels@ferguson.net`
[2] HiFn, `dwhiting@hifn.com`
[3] Counterpane Internet Security, `schneier@counterpane.com`
[4] `kelsey.j@ix.netcom.com`
[5] Universität Mannheim, `lucks@weisskugel.informatik.uni-mannheim.de`
[6] UCSD, `tkohno@cs.ucsd.edu`

Abstract. Helix is a high-speed stream cipher with a built-in MAC functionality. On a Pentium II CPU it is about twice as fast as Rijndael or Twofish, and comparable in speed to RC4. The overhead per encrypted/authenticated message is low, making it suitable for small messages. It is efficient in both hardware and software, and with some pre-computation can effectively switch keys on a per-message basis without additional overhead.

Keywords: Stream cipher, MAC, authentication, encryption.

1 Introduction

Securing data in transmission is the most common real-life cryptographic problem. Basic security services require both encryption and authentication. This is (almost) always done using a symmetric cipher—public-key systems are only used to set up symmetric keys—and a Message Authentication Code (MAC).

The AES process provided a number of very good block cipher designs, as well as a new block cipher standard. The cryptographic community learned a lot during the selection process about the engineering criteria for a good cipher. AES candidates were compared in performance and cost in many different implementation settings. We learned more about the importance of fast rekeying and tiny-memory implementations, the cost of S-boxes and circuit-depth for hardware implementations, the slowness of multiplication on some platforms, and other performance considerations.

The community also learned about the difference of cryptanalysis in theory versus cryptanalysis in practice. Many block cipher modes restrict the types of attack that can be performed on the underlying block cipher. Yet the generally accepted attack model for block ciphers is very liberal. Any method that distinguishes the block cipher from a random permutation is considered an attack. Each block cipher operation must protect against all types of attack. The resulting over-engineering leads to inefficiencies.

Computer network properties like synchronization and error correction have eliminated the traditional synchronization problems of stream-cipher modes like OFB. Furthermore, stream ciphers have different implementation properties that restrict the cryptanalyst. They only receive their inputs once (a key and a nonce) and then produce a long stream of pseudo-random data. A stream cipher can start with a strong cryptographic operation to thoroughly mix the key and nonce into a state, and then use that state and a simpler mixing operation to produce the key stream. If the attacker tries to manipulate the inputs to the cipher he encounters the strong cryptographic operation. Alternatively he can analyse the key stream, but this is a static analysis only. As far as we know, static attacks are much less powerful than dynamic attacks. As there are fewer cryptographic requirements to fulfill, we believe that the key stream generation function can be made significantly faster, per message byte, than a block cipher can be. Given the suitability of steam ciphers for many practical tasks and the potential for faster implementations, we believe that stream ciphers are a fruitful area of research.

Additionally, a stream cipher is often implemented—and from a cryptographic point of view, should always be implemented—together with a MAC. Encryption and authentication go hand in hand, and significant vulnerabilities can result if encryption is implemented without authentication. Outside the cryptographic literature, not using a proper MAC is one of the commonly encountered errors in stream cipher systems. A stream cipher with built-in MAC is much more likely to be used correctly, because it provides a MAC without the associated performance penalties.

Helix is an attempt to combine all these lessons.

2 An Overview of Helix

Helix is a combined stream cipher and MAC function, and directly provides the authenticated encryption functionality. By incorporating the plaintext into the stream cipher state Helix can provide the authentication functionality without extra costs [Gol00].

Helix's design strength is 128 bits, which means that we expect that no attack on the cipher exists that requires fewer than 2^{128} Helix block function evaluations to be carried out. Helix can process data in less than 7 clock cycles per byte on a Pentium II CPU, more than twice as fast as AES.

Helix uses a 256-bit key and a 128-bit nonce. The key is secret, and the nonce is typically public knowledge. Helix is optimised for 32-bit platforms; all operations are on 32-bit words. The only operations used are addition modulo 2^{32}, exclusive or, and rotation by fixed numbers of bits. The design philosophy of Helix can be summarized as "many simple rounds."

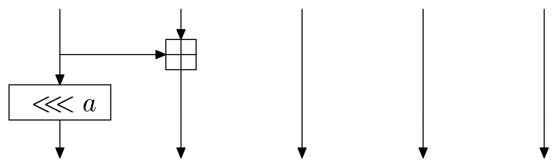

Fig. 1. A single round of Helix

Helix has a state that consists of 5 words of 32 bits each. (This is the maximum state that can fit in the registers of the current Intel CPUs.) A single round of Helix consists of adding (or XORing) one state word into the next, and rotating the first word. This is shown in Figure 1 where the state words are shown as vertical lines. Multiple rounds are applied in a cyclical pattern to the state. The horizontal lines of the rounds wind themselves in helical fashion through the five state words. Twenty rounds make up one block (see Figure 2). Helix actually uses two intertwined helices; a single block contains two full turns of each of the helices.

During each block several other activities occur. During block i one word of key stream is generated (S_i), two words of key material are added ($X_{i,0}$ and $X_{i,1}$), and one word of plaintext is added (P_i). The output state of one block is used as input to the next, so the computations shown in figure 2 are all that is required to process 4 bytes of the message. As with any stream cipher, the ciphertext is created by XORing the plaintext with the key stream (not shown in the figure).

At the start of an encryption a starting state is derived from the key and nonce. The key words $X_{i,j}$ depend on the key, the length of the input key, the nonce, and the block number i. State guessing attacks are made more difficult by adding key material at double the rate at which key stream material is extracted. At the end of the message some extra processing is done, after which a 128-bit MAC tag is produced to authenticate the message.

3 Definition of Helix

The Helix encryption function takes as input a variable length key U of up to 32 bytes, a 16-byte nonce N, and a plaintext P. It produces a ciphertext message and a tag that provides authentication. The decryption function takes the key, nonce, ciphertext, and tag, and produces either the plaintext message or an error if the authentication failed.

3.1 Preliminaries

Helix operates on 32-bit words while the inputs and outputs are a sequences of bytes. In all situations Helix uses the least-significant-byte first convention. A

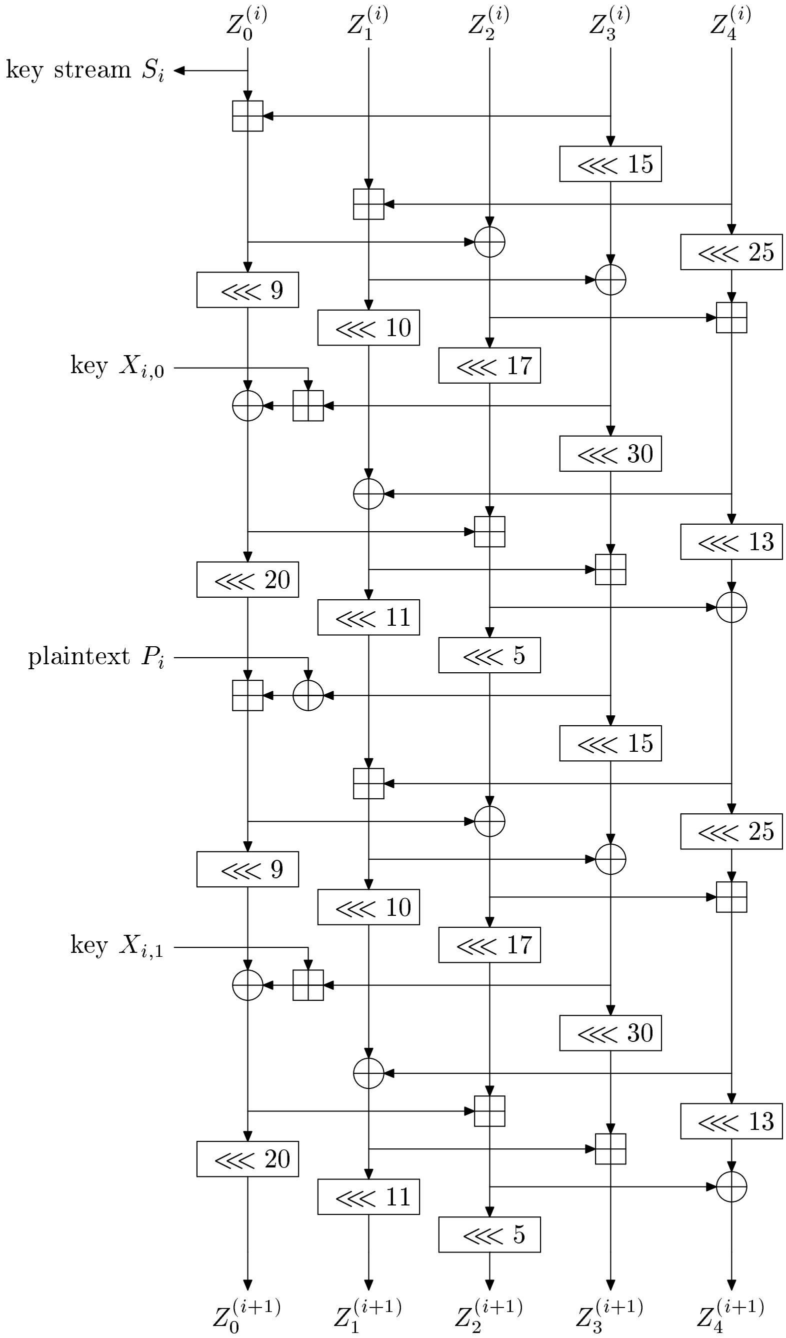

Fig. 2. One block of Helix encryption

sequence of bytes x_i is identified with a sequence of words X_j by the relations

$$X_j := \sum_{k=0}^{3} x_{(4j+k)} \cdot 2^{8k} \qquad\qquad x_i := \left\lfloor \frac{X_{\lfloor i/4 \rfloor}}{2^{8(i \bmod 4)}} \right\rfloor \bmod 2^8$$

These two equations are complimentary and show the conversion both ways.

Let $\ell(x)$ denote the length of a string of bytes x. The input key U consists of a sequence of bytes $u_0, u_1, \ldots, u_{\ell(U)-1}$ with $0 \leq \ell(U) \leq 32$. The key is processed through the key mixing function, defined in section 3.7, to produce the working key which consists of 8 words $K_0, \ldots, K_7$.

The nonce N consists of 16 bytes, interpreted as 4 words $N_0, \ldots, N_3$.

The plaintext P and ciphertext C are both sequences of bytes of the same length, with the restriction that $0 \leq \ell(P) < 2^{64}$. Both are manipulated as a sequence of words, P_i and C_i respectively. The last word of the plaintext and ciphertext might be only partially used. The 'extra' plaintext bytes in the last word are taken to be zero. The 'extra' ciphertext bytes are irrelevant and never used. Note that the cipher is specified for zero-length plaintexts; in this case, only a MAC is generated.

3.2 A Block

Helix consists of a sequence of blocks. The blocks are numbered sequentially which assigns each block a unique number i. At the start of block i the state consists of 5 words: $Z_0^{(i)}, \ldots, Z_4^{(i)}$; at the end of the block the state consists of $Z_0^{(i+1)}, \ldots, Z_4^{(i+1)}$ which form the input to the next block with number $i+1$. Block i also uses as input two key words $X_{i,0}$ and $X_{i,1}$, and the plaintext word P_i. It produces one word of key stream $S_i := Z_0^{(i)}$; the ciphertext words are defined by $C_i := P_i \oplus S_i$.

Instead of repeating the block definition in formulas, we define the block function using figure 2. All values are 32-bit words, exclusive or is denoted by $\oplus$, addition modulo 2^{32} is denoted by $\boxplus$, and rotation by $\lll$.

In the remainder of this paper, the terms "block," and "block function" are used interchangeably.

3.3 Key Words for Each Block

The expanded key words are derived from the working key $K_0, \ldots, K_7$, the nonce $N_0, \ldots, N_3$, the input key length $\ell(U)$, and the block number i. We first extend the nonce to 8 words by defining $N_k := (k \bmod 4) - N_{k-4} \pmod{2^{32}}$

for $k = 4, \ldots, 7$. The key words for block i are then defined by

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

where all additions are taken modulo 2^{32}. Note that X'_i encodes bits 31 to 62 of the value $i + 8$; this is not the same as the upper 32 bits of $i + 8$.

3.4 Initialisation

A Helix encryption is started by setting

$$Z_i^{(-8)} = K_{i+3} \oplus N_i \quad \text{for } i = 0, \ldots, 3$$
$$Z_4^{(-8)} = K_7$$

Eight blocks are then applied, using block number -8 to -1. For these block the plaintext word P_i is defined to be zero, and the generated key stream words are discarded.

3.5 Encryption

After the initialisation the plaintext is encrypted. Let $k := \lfloor (\ell(P) + 3)/4 \rfloor$ be the number of words in the plaintext. The encryption consists of k blocks numbered 0 to $k - 1$. Each block generates one word of key stream, which is used to encrypt one word of the plaintext. Depending on $\ell(P) \bmod 4$, between 1 and 4 of the bytes of the last key stream word are used.

3.6 Computing the MAC

Just after the block that encrypted the last plaintext byte, one of the state words is modified. The state word $Z_0^{(k)}$ is XORed with the value 0x912d94f1.[1] Using this modified state, eight blocks, numbered $k, \ldots, k + 7$ are applied for post-mixing. For these blocks the generated key stream is discarded and the plaintext word P_i is defined as $\ell(P) \bmod 4$. After the post-mixing, four more blocks, numbered $k + 8, \ldots, k + 11$, are applied. The key stream generated by these four blocks form the tag. The plaintext input remains the same as in the previous eight blocks.

[1] This constant is constructed by taking the 6 least significant bits of each of the ASCII characters of the string "Helix", and putting a single one bit both before and after it.

3.7 Key mixing

The key mixing converts a variable-length input key U to the fixed-length working key, K.

First, the Helix block function is used to create a round function F that maps 128 bits to 128 bits. The four input words to F are extended with a single word with value $\ell(U) + 64$ to form a 5-word state. The block function is then applied with zero key inputs and zero plaintext input. The first four state words of the resulting state form the result of F.

The input key U is first extended with $32 - \ell(U)$ zero bytes. The 32 key bytes are converted to 8 words $K_{32}, \ldots, K_{39}$. Further key words are defined by the equation

$$(K_{4i}, \ldots, K_{4i+3}) := F((K_{4i+4}, \ldots, K_{4i+7})) \oplus (K_{4i+8}, \ldots, K_{4i+11})$$

for $i = 7, \ldots, 0$. The words $K_0, \ldots, K_7$ form the working key of the cipher. (This recursion defines a Feistel-type cipher on 256-bit blocks.)

3.8 Decryption

Decryption is almost identical to encryption. The only differences are:

- The key stream generated at the start of each block is used to decrypt the ciphertext and produce the plaintext word that is required half a block later. Care has to be taken with the last plaintext word to ensure that unused plaintext bytes are taken to be zero and not filled with the extra key stream bytes.
- Once the tag has been generated it is compared to the tag provided. If the two values are not identical, all generated data (i.e. the key stream, plaintext, and tag) is destroyed.

4 Implementation

Compared to other ciphers Helix is relatively easy to implement in software. If 32-bit addition, exclusive or, and rotation functions are available, all the functions are easily implemented. Helix is also fast. A single round takes only a single clock cycle to compute on a Pentium II CPU, because the super-scalar architecture can perform an addition or XOR simultaneously with a 32-bit rotation. A block of Helix takes 20 cycles plus some overhead for the handling of the plaintext, key stream, and ciphertext. Our un-optimised assembly implementation requires less than 7 clock cycles per byte. This compares to about 16 clock cycles per byte for the best AES implementation on the same platform.[2]

[2] This is a somewhat unfair comparison. The AES implementation does not actually read the data from memory, encrypt it, and write it back, which would slow it down further. What is more, most block cipher modes only provide encryption *or* authentication so two passes over the message are required. The alternative is to use one of the new authenticated encryption modes, such as [Jut01], but they are all patented and require a license.

Most implementation flexibility is in the way the key schedule is computed. The key mixing only needs to be done once for each key value. The recurrence relation used in the key mixing implements a Feistel cipher, so the key mixing can be done in-place. The $X_{i,1}$ key words can mostly pre-computed with only the block number being added every block. Implementations that limit the plaintext size to 2^{32} bytes can ignore the upper bits of the block number in the definition of X'_i because these bits will always be zero.

Helix is also fast in hardware. The rotations cost no time, although they do consume routing resources in chip layouts. The critical path through the block function consists of 6 additions and 5 XORs. As the critical path contains no rotations, a certain amount of ripple of the adders can be overlapped, with the lower bits being produced and used before the upper bits are available. A more detailed analysis of this overlapping is required for any high-speed implementation. A conservative estimate for a relatively low-cost ASIC layout is 2.5 ns per 32-bit adder and 0.5 ns per XOR , which adds up to 17.5 ns/block. This translates to more than 200 MByte per second, or just under 2 Gbit per second.

5 Use

One of the dangers of a steam cipher is that the key-stream will be re-used. To avoid this problem Helix imposes a few restrictions on the sender and receiver:

- The sender *must* ensure that each (K,N) pair is used at most once to encrypt a message. A single sender must use a new, unique, nonce for each message. Multiple senders that want to use the same key have to ensure that they never choose the same nonce, for example by dividing the nonce space between them. If two different messages are ever encrypted with the same (K,N) pair, Helix loses its security properties.
- The receiver may not release the plaintext P, or the key stream, until it has verified the tag successfully. In most situations this requires the receiver to buffer the entire plaintext before it is released.

These requirements seem restrictive, but they are in fact implicitly required by all stream ciphers (e.g. RC4) and many block cipher modes (e.g. OCB [RBBK01b,RBBK01a] and CCM [WHF])

Although Helix allows the use of short keys, we strongly recommend the use of keys of at least 128 bits, preferably 256 bits.

6 Other modes of use

So far we have described Helix as providing both encryption and authentication. Helix can be used in other modes as well. For any particular key Helix should be used in only one of these modes. Using several modes with a single key can lead to a loss of security.

6.1 Unencrypted Headers

In packet environments it is often desirable to authenticate the packet header without encrypting it. From the encryption/authentication layer this looks like an additional string of data that is to be authenticated but not encrypted. We define a standard method of handling such additional data without modifying the basic Helix computations.

First a length field is formed which is eight bytes long and encodes the length of the additional data in least-significant-byte first format. The additional data is padded with 0–3 zero bytes until the length is a multiple of four. The concatenation of the length field, the padded additional data, and the message data are then processed as a normal message through Helix. The ciphertext bytes corresponding to the length field and the padded additional data are discarded, leaving only the ciphertext of the message data and the tag.

6.2 Pure stream cipher & PRNG

Helix can be use as a pure stream cipher by ignoring the MAC computations at the end. And like any stream cipher, Helix is a cryptographically strong pseudo-random number generator. For every (key,nonce) input it produces a stream of pseudo-random data. This makes Helix suitable for use as a PRNG.

6.3 MAC with Nonce

Helix can also be used a pure MAC function. The data to be authenticated is encrypted, but the ciphertext is discarded. The receiver similarly discards the key stream and just feeds the plaintext to the Helix rounds. In this mode Helix is significantly faster than, for example, HMAC-SHA1, but it does require a unique nonce for each message. Unfortunately, it is insecure to use Helix with a fixed nonce value, due to collisions on the 160-bit state.

7 Design rationale

Although the design strength of Helix is 128 bits, we use 256-bit keys. This avoids a very general class of attacks that exploits collisions on the key value. For flexibility Helix also allows shorter keys to be used, as there are many practical situations in which fewer than 256 bits of key material are available.

The small set of elementary operations that Helix uses makes it efficient on a large number of platforms. The absence of tables makes Helix efficient in hardware as well.

Most ciphers use lookup tables to provide the necessary nonlinearity. In Helix the nonlinearity comes from the mixing of XORs with additions. Neither of these operations can be approximated well within the group of the other.

There are some good approximations, but on average the approximations are quite bad [LM01].

The diffusion in Helix is not terribly fast, but it is unstoppable. As 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 to ensure thorough mixing of the state words.

The key mixing is an un-keyed bijective function. The purpose is to spread the available entropy over all key words. If, for example, the key is provided by a SHA-1 computation then only 5 words of key material are provided. The key mixing ensures that all 8 key words depend on the key material. Using a bijective mixing function ensures that no two 256-bit input keys lead to the same working key values. The use of the input key length in X' guarantees that even keys that lead to the same working key (each short key leads to a working key that is also produced by a 256-bit key) do not lead to equivalent Helix encryptions.

7.1 Key schedule

The $X_{i,0}$ values simply cycle through the key words. The $X_{i,1}$ values depend on the same key words in anti-phase, the extended nonce words, the block number, and the input key length. This key schedule has a number of properties. All 8 key words and and all 4 nonce words affect the state every 4 blocks.

The key schedule also ensures that different (K, N) pairs produce different block key sequences. Even stronger: no sequence of 17 key words ever occurs twice across all keys, all nonce values, and all positions in the encryption computation.

To demonstrate this we look at the sequence $Y_j := X_{\lfloor j/2 \rfloor, j \bmod 2}$. This is the sequence of key words in the order they are used. Given just part of the sequence Y_j, without the proper index values j, we can recover the key, nonce, and block number. (When the plaintext word is zero the first half of the block function is identical to the second half of the block function, so it makes sense to look at the sequence Y_j and allow half-block offsets.)

If $Y_j = Y_{j+16}$ then j is even, otherwise j is odd. This allows us to split the Y values back into an $X_{i,0}$ and $X_{i,1}$ sequence.

Now consider

$$\begin{aligned} R_i &:= X_{i,1} - X_{i,0} + X_{i+4,1} - X_{i+4,0} \\ &= N_{i \bmod 8} + N_{(i+4) \bmod 8} + X'_i + X'_{i+4} + 2i + 20 \\ &= (i \bmod 4) + 2i + 20 + X'_i + X'_{i+4} \end{aligned}$$

all modulo 2^{32}. We first look at $R_i \bmod 4$. The X' terms can only have a nonzero contribution if $i \bmod 4 = 3$, so 3 out of 4 consecutive times we get just $((i \bmod 4) + 2i) \bmod 4 = 3i \bmod 4$, which gives us $i \bmod 4$. Looking at the full R_i for an i with $i \bmod 4 = 0$ gives us $i \bmod 2^{31}$. The sum $X'_i + X'_{i+4}$ from the case $i \bmod 4 = 3$ gives us the upper bits of i. This recovers[3] the block number, i.

Given $i \bmod 8$ we can recover the working key from the $X_{i,0}$'s. Knowledge of i and the key words allows us to compute the key length and the nonce from the $X_{i,1}$'s, as well as check the redundancy introduced by the nonce expansion to 8 words.

We have not investigated whether it is possible to recover the key, nonce, and block number from fewer than 17 consecutive key words. A simple counting argument shows that at least 14 are required. This remains an open problem.

7.2 Choice of Rotation Counts

The strength of Helix is depends on the rotation counts chosen for the Helix block function. The rotations provide the diffusion between the various bit positions in the state words. During the design process we examined the impact of various choices of rotation counts both in terms of attempts to cryptanalyze the cipher, and also in terms of their impact on statistical tests of the block function.

To analyse the diffusion properties of a set of rotation counts, consider a variant of the block function with all the additions are changed to XORs. (This is equivalent to ignoring the carries in the additions.) In this variant we can track which output bits are affected by which input bits. For this analysis we consider an output bit affected if its computational path has a dependency on the input bit at any one point, even if the output bit in our linearised block function is not changed due to several dependencies canceling out. This seems to be the most suitable way to analyse diffusion and is related to the independence assumption in differential and linear cryptanalysis.

A set of rotation counts can, at best, ensure that changing a single state input bit affects at least 21 bits of the output. There are a large number (over 6 000) of such rotation count sets.

We discarded all rotation count sets that contained a rotation count of 0, 1, 8, 16, 24, or 31. Rotation by a multiple of 8 has a relatively low order, and rotation by 1 or 31 bit positions provides diffusion between adjacent bits, something the carry bits already do. This reduced the set of candidate rotation counts to 86.

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 160-bit internal state. Among our tests, we considered:

[3] This isn't absolutely perfect. We don't recover the 62'nd bit of $i + 8$, but this bit will only be set during the very last few blocks of a message very close to 2^{64} bytes long. This does not lead to a weakness.

1. The number of rounds required before all output bits passed binomial tests given a fixed input difference in the state.
2. The number of rounds required before the output states' Hamming weight distribution passed a χ^2 test given low- and high-Hamming weight input states.
3. The number of round required before the output states' differences Hamming weight distribution passed a χ^2 test given low- and high-Hamming weight differences in the input state [KRRR98].
4. Low- and high-Hamming weight higher-order differences, and the number of rounds required before the resulting output differences' Hamming weights passed a χ^2 test.

The surprising result was that most rotation counts did pretty well. Our carefully-selected rotation count sets were slightly better than random ones, but only by a small margin. Degenerate rotation counts (all rotation counts equal, or most rotation counts zero) led to much worse test results.

At the end of our analysis, we selected more or less at random from the remaining candidates. Based on our limited analysis, the specific choice of rotation counts does not have a strong impact on the security of Helix, with only the caveat that we had to avoid some obvious degenerate cases.

8 Conclusions and intellectual property statement

Most applications that require symmetric cryptography actually require both encryption and authentication. We believe that the most efficient way to achieve this combined goal is to design cryptographic primitives specifically for the task. Towards this end, we present such a new cryptographic primitive, called Helix. We hope that Helix and this paper will spur additional research in authenticated encryption stream ciphers. As with any experimental design, we remark that Helix should not be used until it has received additional cryptanalysis.

Finally, we hereby explicitly release any intellectual property rights to Helix into the public domain. Furthermore, we are not aware of any patent or patent application anywhere in the world that cover Helix.

9 Acknowledgements

We would like to thank David Wagner, Rich Schroeppel, and the anonymous referees for their helpful comments and encouragements. Felix Schleer helped us by creating one of the reference implementations.

References

[Arm02] Frederik Armknecht. A linearization attack on the Bluetooth key stream generator. Cryptology ePrint Archive, Report 2002/191, 2002. `http://eprint.iacr.org/2002/191`.

[Cou02] Nicolas Courtois. Higher order correlation attacks, XL algorithm, and cryptanalysis of Toyocrypt. In *Information Security and Cryptology–Icisc 2002*, volume 2587 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002. To appear.

[CP02] Nicolas Courtois and Josef Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In Yuliang Zheng, editor, *Advances in Cryptology—ASIACRYPT2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Springer-Verlag, 2002.

[DGV93] Joan Daemen, René Govaerts, and Joos Vandewalle. Resynchronisation weaknesses in synchronous stream ciphers. In Tor Helleseth, editor, *Advances in Cryptology—EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 159–167. Springer-Verlag, 1993.

[Gol00] Jovan Dj. Golić. Modes of operation of stream ciphers. In Douglas R. Stinson and Stafford Tavares, editors, *Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000*, volume 2012 of *Lecture Notes in Computer Science*, pages 233–247. Springer-Verlag, 2000.

[Jut01] Charanjit S. Jutla. Encryption modes with almost free message integrity. In Birgit Pfitzmann, editor, *Advances in Cryptology—EUROCRYPT2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 529–544, 2001.

[KRRR98] Lars R. Knudsen, Vincent Rijmen, Ronald L. Rivest, and M.J.B. Robshaw. On the design and security of RC2. In Serge Vaudenay, editor, *Fast Software Encryption, 5th International Workshop, FSE'98*, volume 1372 of *Lecture Notes in Computer Science*, pages 206–221. Springer-Verlag, 1998.

[LM01] Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. In Mitsuru Matsui, editor, *Fast Software Encryption2001*, Lecture Notes in Computer Science. Springer-Verlag, To appear, 2001. Available from `http://www.tcs.hut.fi/~helger/papers/lm01/`.

[RBBK01a] Philip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption, September 2001. Available from `http://www.cs.ucdavis.edu/~rogaway`.

[RBBK01b] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A block-cipher mode of operation for efficient authenticated encryption. In *Eighth ACM Conference on Computer and Communications Security (CCS-8)*, pages 196–205. ACM Press, 2001.

[WHF] Doug Whiting, Russ Housley, and Niels Ferguson. Counter with CBC-MAC (CCM). Available from `csrc.nist.gov/encryption/modes/proposedmodes/ccm/ccm.pdf`.

A Test vectors

The authors will maintain a web site at `http://www.macfergus.com/helix` with news, example code, and test vectors. We give some simple test vectors here. (The 8-word working key is given as a sequence of 32 bytes, least significant byte first.)

```
Initial Key: <empty string>
Nonce:       00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Working Key: a9 3b 6e 32 bc 23 4f 6c 32 6c 0f 82 74 ff a2 41
             e3 da 57 7d ef 7c 1b 64 af 78 7c 38 dc ef e3 de
Plaintext:   00 00 00 00 00 00 00 00 00 00
Ciphertext:  70 44 c9 be 48 ae 89 22 66 e4
```

```
MAC:          65 be 7a 60 fd 3b 8a 5e 31 61 80 80 56 32 d8 10

Initial Key: 00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00
             04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00
Nonce:       00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00
Working Key: 6e e9 a7 6c bd 0b f6 20 a6 d9 b7 59 49 d3 39 95
             04 f8 4a d6 83 12 f9 06 ed d1 a6 98 9e c8 9d 45
Plaintext:   00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00
             04 00 00 00 05 00 00 00 06 00 00 00 07 00 00 00
Ciphertext:  7a 72 a7 5b 62 50 38 0b 69 75 1c d1 28 30 8d 9a
             0c 74 46 a3 bf 3f 99 e6 65 56 b9 c1 18 ca 7d 87
MAC:         e4 e5 49 01 c5 0b 34 e7 80 c0 9c 39 b1 09 a1 17

Initial Key: 48 65 6c 69 78
Nonce:       30 31 32 33 34 35 36 37 38 39 61 62 63 64 65 66
Working Key: 6c 1e d7 7a cb a3 a1 d2 8f 1c d6 20 6d f1 15 da
             f4 03 28 4a 73 9b b6 9f 35 7a 85 f5 51 32 11 39
Plaintext:   48 65 6c 6c 6f 2c 20 77 6f 72 6c 64 21
Ciphertext:  6c 4c 27 b9 7a 82 a0 c5 80 2c 23 f2 0d
MAC:         6c 82 d1 aa 3b 90 5f 12 f1 44 3f a7 f6 a1 01 d2
```

B Cryptanalysis

Helix is intended to provide everything needed for an encrypted and authenticated communications session. A successful attack on Helix will have occurred when an attacker can either predict a keysteam bit he hasn't seen with a probability slightly higher than 50%, or when he can create a forged or altered message that is accepted by the recipient with a probability substantially higher than 2^{-128}. To be meaningful given the 128-bit security bound of Helix, any such attack must require fewer than 2^{128} block function evaluations for all participants combined. Also, any such attack must obey the security requirements placed on Helix' operations, e.g., no reuse of nonces, MACs checked before decrypted messages released, etc.

In this section, we consider a number of possible ways to attack Helix. Although our time and resources have been limited, we have not yet discovered any workable method of attacking Helix.

B.1 Static analysis

A static analysis just takes the key stream 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 key stream words are fully random. This is because each $X_{i,1}$ key value affects S_{i+1} in a bijective manner, so for any given state and any sequence of $X_{i,0}$ words there is a bijective mapping from $K_{(i+4) \bmod 8}, \ldots, K_{(i+7) \bmod 8}$ to $S_{i+1}, \ldots, S_{i+4}$. 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 therefore span at least 4 blocks and 5 key stream words. Of course, there is no reasonable way of finding the state. At the beginning of each block there is 128 bits of unknown state. (The 32 bits of the key stream word are known to the attacker.) As the design strength is 128 bits, an attacker cannot afford to guess the entire state. A partially guessed state does not help much as key material is added at twice the rate that key stream is produced.

B.2 Period length

Helix' internal state is updated continuously by the plaintext it is encrypting. So long as the plaintext is not repeating, the keystream should have an arbitrarily long period.

With a fixed or repeating plaintext, the Helix state does not cycle either. In section 7.1 we showed that any 17 consecutive key words used as inputs to the block function are unique. The nonrepeating key word values prevent the state from ever falling into a cycle.

B.3 State collisions

The 160-bit state of Helix can be expected to collide for some (key,nonce) pairs. However, this doesn't lead to a weakness, because the state collision is guaranteed not to survive long enough to yield an attack, or even allow reliable detection by the attacker.

To detect a collision on 160 bit values requires 160-bits of information about each state. But in the four block computations required to generate 160 bits of key stream the whole key, nonce, and block number get added to the state. Starting at the same state these inputs will introduce a difference in the key stream, and make it impossible to detect the state collision[4].

B.4 Weak keys

Helix makes constant use of the words of the working key. An all-zero working key intuitively seems like a bad thing (it effectively omits a few operations from the block function), but we have not discovered any possible attack based on it. The all-zero working key is only generated by a single key of 32 bytes length. Shorter key length cannot generate the all-zero working key. The all-zero working key does not seem to have any practical security relevance, and there is no reason to treat this key differently from any other key.

[4] State collisions where the key and nonce are the same and the block number differs only in the upper 30 bits also do not lead to an attack.

B.5 Adaptive chosen plaintext attacks

Because the plaintext affects the state, Helix allows an attack model that traditional stream ciphers prevent: An attacker can request the encryption of a plaintext block under an already established (key,nonce) pair, and can use the resulting ciphertext to determine what plaintext to request next.

We have found no way to use such an attack against Helix. As with the discussion of static analysis, above, the large unknown and untouchable state, and the continual mixing of key material into that state, appear to defeat attempts to use control over one input of the block function to control other parts of its state. Additionally, the usage restrictions on Helix do not allow reuse of nonces, which ensures that the state is always a "moving target."

B.6 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 Helix, this can be done only with the key or the nonce. In each case, the block function is applied multiple times to the input. In Helix all the places where such attacks are possible we have eight consecutive blocks without any output. A change to the nonce, such as is considered in [DGV93], will be thoroughly mixed into the state by the time the first key stream word is generated. Similarly, a change to the last plaintext byte is thoroughly mixed into the state before the first MAC tag word is generated. A differential attack would have to use a differential through 8 blocks, or 160 rounds of Helix. A search found no useful differentials for 8 blocks of Helix, nor useful higher-order differentials.

B.7 Algebraic attacks over GF(2)

The only reasonable line of attack we have found so far is to apply equation-solving techniques. In 2002, XSL was used to analyse block ciphers [CP02]. An attack on Serpent seems to be marginally better than brute force, another attack on the AES is slower than brute force. Similar techniques have been used to successfully analyse stream ciphers [Cou02,Arm02].

We have tried to analyse Helix by algebraic techniques. Under an optimistic assumption (from the attacker's point of view) on the number of linear-independent equations, the best attack we could think of requires solving an (overdefined) system of $\approx 2^{49.7}$ linear equations in $N = 2^{49.1}$ binary variables. Gaussian elimination needs $N^3 \approx 2^{147.3}$ steps, and falls well outside our security bound.

[CP02] suggest to use another algorithm, which takes $O(N^{2.376})$ steps, but with an apparently huge proportional constant. In our case $N^{2.376} \approx 2^{116.7}$, so even a relatively small proportional constant pushes this beyond our security bound.[5] Our analysis has not resulted in an attack that requires less work than 2^{128} block function evaluations, and we conjecture that no such attack exists.

C Single Helix

Most ciphers are analysed by first creating simplified versions and attacking those. Apart from the obvious methods of simplifying Helix we present Single Helix as an object for study. Single Helix uses only one helix instead of two interleaved ones, and has significantly slower diffusion in the backwards direction. A block of single Helix is shown in Figure 3. This uses an alternative configuration where the key and plaintext inputs are added directly to the state words.

[5] Due to space constraints, we left out a more detailed description of the attack.

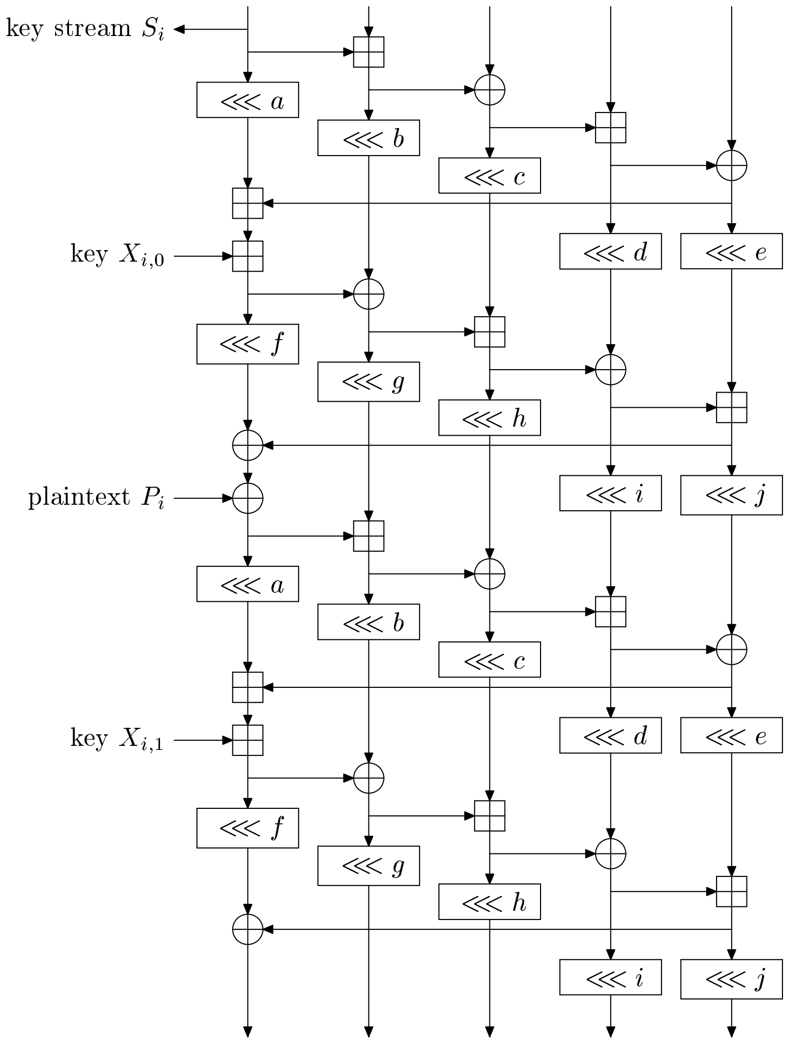

Fig. 3. A round of Single-Helix