The HKC authenticated stream cipher (Ver.1)

Matt Henricksen[1], Shinsaku Kiyomoto[2], and Jiqiang Lu[1]

[1] Institute for Infocomm Research
1 Fusionopolis Way 21-01 Connexis (South Tower), 138632, Singapore
mhenricksen@i2r.a-star.edu.sg
[2] KDDI R & D Laboratories Inc.
2-1-15 Ohara, Fujimino-shi, Saitama, 356-8502, Japan
kiyomoto@kddilabs.jp

Date:14/03/2014

1 Introduction

Authenticated encryption provides confidentiality and integrity in the same one-pass process, by computing a Message Authentication Code (MAC) at almost no cost over generating keystream. This is natively more efficient than providing confidentiality then authentication in separate passes. The argument as to whether compute MAC (on plaintext) then encrypt, encrypt then compute MAC (on ciphertext), or simultaneously encrypt and compute MAC (on plaintext, then encrypt plaintext and append MAC to ciphertext) was settled by Bellare and Namprempre [1] who showed that computing a MAC of the ciphertext best satisfies security properties of the typical scheme.

HKC is a new stream cipher with a built in MAC routine that provides 'encrypt then MAC'. HKC uses a 256-bit key to provide 256 bits of security. Its native environment is a processor that operates on 64-bit words, although it is also efficient in other environments. HKC is a family of **Stream-Cipher-Based Authenticated Encryption** algorithms.

HKC's heritage is the successful eSTREAM software portfolio cipher HC[2], designed by Hongjun Wu. HC is defined in two versions, respectively with 128- and 256-bit keys, although only the former is included in the eSTREAM portfolio. HKC is based upon HC-256 to incorporate a MAC, to take advantage of larger register sizes on modern processors, and to remove or diminish what HKC's authors perceive as inefficiencies.

2 Security Goals

The design strength of HKC is 256 bits, which means that no attack to recovery the key or forge a MAC should exist that requires significantly less time than it takes to test 2^{256} candidates for a secret key. HKC uses a secret 256-bit key (K) and a public 256-bit initialization vector (IV). The maximum length of keystream which is generated from the same key-IV pair is limited to less than 2^{70} bits. For HKC, we do not assume that key-IV pairs are replayed.

Table 1. Security Goals

Category	Security Level
Confidentiality for the plaintext	256-bit level
Integrity for the plaintext	256-bit level
Integrity for the associated data	256-bit level

3 Specification

The state of HKC comprises a table W, which consists of 512 elements, a register M containing four elements, and a 64-bit counter i. W is used primarily for keystream generation, and M for MAC accumulation. Each element in W and M, is a 64-bit word, where word x can be denoted as a series of bytes $x = x_0||x_1||...||x_7$, from least- to most-significant byte respectively.

There are at least three phases during the execution of a HKC session - initialization, keystream generation/MAC accumulation, and MAC generation. An optional fourth phase - incorporation of unencrypted associated data in the MAC - is discussed in Section 3.4.

At the start of the session, the state of HKC is initialized using a secret 256-bit key (K) and a public 256-bit initialization vector (IV). It is not permitted to initialize HKC with the same key-IV pair more than once; if not complied, then the guarantees about security do not hold. For $i \geq 2^{64}$, re-initialization with an updated IV is required. After HKC is primed, it is clocked to generate 64-bit words of keystream, z_i at time i. Finally, at the end of the the session, MAC generation is used to compute a 256-bit MAC, which represents a fingerprint of all of the ciphertexts to have been encrypted during the session, and any associated data.

Each of these phases uses a different combination of sub-functions, as described below.

$f(x)$ is used only in key initialization.

$$f(x) = (x \ggg 7) \oplus (x \ggg 47) \oplus (x >> 3)$$

$g(x, y, z)$ is the update function during keystream generation and key initialization. Parameter $z \in \{0, 1\}$ determines W_0 or W_1, where W_0 is the table W constrained to elements $0 \cdots 255$, and W_1 is the table W constrained to elements $256 \cdots 511$.

$$g(x, y, z) = ((x \ggg 10) \oplus (y \ggg 35)) + W_z[(x \oplus y) \bmod 256]$$

$h(x)$ is the output filter used during keystream generation.

$$h(x) = W[256 + x_7] + W[128 + x_4] + W[x_1]$$

3.1 Initialization process

1. Let $K = K_0||K_1||K_2||K_3$ and $IV = IV_0||IV_1||IV_2||IV_3$, where K_i and IV_i each denote a 64-bit quantity. They are expanded into the 512-element table W as:

$$W[i] = \begin{cases} K_i & 0 \le i < 4 \\ IV_{i-4} & 4 \le i < 8 \\ f(W[i-1]) + f(W[i-8]) + W[i-3] + i & 8 \le i < 512 \end{cases}$$

2. The table M is initialized as:

$$M[i] = W[512+i] \text{ for } -4 < i < 0$$
$$M[i] = f(M[i-1]) + f(W[i+504]) + M[i-3] + (512+i) \text{ for } 0 \le i < 4$$

3. Set the counter i to 0. Clock the cipher 512 steps, according to Section 3.2, but discarding keystream. In this phase, the MAC register M is not updated. The counter value ends at 512.

The initialization process is complete, and the cipher ready to generate output.

3.2 Keystream generation and message accumulation

At each step during keystream generation, one element of the table is updated, one keystream word is output and used to encrypt plaintext or decrypt ciphertext, and the MAC register is updated. All references to W in this section are modulo 512.

Encrypting/decrypting algorithms The following process is run to generate keystream word z_i.

$$W[i] = W[i] + W[i-15] + g(W[i-4], W[i+1], \zeta(i))$$
$$z_i = h(W[i-13]) \oplus W[i];$$

where $\zeta(i) = (i >> 8) \oplus 1$ - ie. it selects the half of the table not indicated by i. The value of the index i is the counter mod 512, which is incremented once for every invocation of g.

The ciphertext C_i is formed from plaintext P_i and the keystream word using binary addition - ie. $C_i = P_i \oplus z_i$. Conversely, the ciphertext is decrypted to plaintext as $P_i = C_i \oplus z_i$.

Let r be the number of 64-bit words in the message encrypted in the session. Then for the number of plaintext bytes $\ell(P), 8(r-1) < \ell(P) \le 8r$. If $\ell(P)$ is not a multiple of eight, then it is up to the protocol that uses HKC how to treat the redundant bytes in the last word of plaintext or ciphertext.

Two examples are to use null bytes; eg. for $\ell(P) = 1$, the block is $xx|00|..|00$; another is to use SHA-style padding. eg. for $\ell(P) = 1$, the block is $xx|80|00|..|01$.

Computing MAC Once the keystream word has been generated, and one element of W updated, the MAC registers M are modified using table W and ciphertext C_i as

$$M[j] = M[j+1], 0 \leq j < 3$$
$$M[3] = (M[0] \oplus M[1] \oplus W[M[3] \bmod 512]) + C$$

3.3 MAC generation

After the final keystream word C_r is generated, the MAC register is modified along with the first sixteen elements of W (ie. in this section, all indices of W are mod 16).

Firstly, the last word of M is modified by the number of bytes of plaintext encrypted in this session:

$$M[3] = M[3] \oplus \ell(P) \text{ where } 8(r-1) < \ell(P) \leq 8r$$

Both W and M are clocked sixteen times, $0 \leq s < 16$, according to the following rules:

The ciphertext word for the iteration is updated as

$$C = C + W[s]$$

$$W[s] = W[s] + g_1(M[3], f(C) \oplus s), 0 \leq s < 16$$

where

$$g_1(x, y) = ((x \ggg 10) \oplus (y \ggg 35)) + W[(x \oplus y) mod 16]$$

followed by

$$M[i] = M[i+1], 0 \leq i < 3$$
$$M[3] = (M[0] \oplus M[1] \oplus W[M[3] \bmod 16]) + C$$

Note that during this stage, no ciphertext is released.

The 256-bit MAC is the contents of the register M at the end of this phase - ie. $M[0]|M[1]|M[2]|M[3]$. If a smaller MAC is desired, then it can be formed by truncation.

3.4 Associated data

HKC permits the inclusion of unencrypted associated header data in the computation of the MAC tag. It can process a message $H||P$, where H is a packet header, and P the unencrypted body of the packet, to produce $H||C||T$, where C is the encrypted packet body, H is the unaltered header, and T is the MAC tag that represents both H and C. If the value of IV is required to be authenticated, the value is included in H.

There is only one caveat - the header H must be padded such that its length in bytes - $\ell(H)$ - is a multiple of eight.

The header is encrypted as per normal, but the ciphertext produced by the cipher is discarded. When the final block of the header has been encrypted, the MAC register value $M[3]$ is modified by xoring with the 64-bit constant $\ell(H)$. Then processing of the plaintext into ciphertext proceeds. If no associated ata is processed, then $\ell(H) = 0$.

4 Design Rational

The structure of HKC is similar to that of HC-256, except that the word size has been increased, and the number and size of tables reduced. In the nine years since HC-256, which operates on 32-bit words, was published, a significant number of general purpose machines now operate on 64-bit registers. HKC takes advantage of this by doing nothing other than doubling the word size, the throughput of the cipher is also doubled. However, other changes are necessary to improve the security, since doubling word-size increases the leak of information from the state.

Compared to many other ciphers, the ratio of key size to state size in HC is very large. As examples, RC4 with a 256-bit key has a ratio of 1:8, Dragon with a 256-bit key has a ratio of 1:4.25, Rabbit with a 256-bit key has a ratio of 1:4.01, and Sosemanuk with a 256-bit key has a ratio of 1:1.5. Comparatively, HC-256 has an extremely large ratio of 1:256. There are no effective attacks on HC-256, or HC-128, which has a half-size state, and no indication that there are effective attacks against a smaller version of HC. Conversely, such a large state size is inconvenient during the initialization stage, which invokes the update function twice on every element of both P and Q. This means that initialization in HC-256 takes around 75,000 cycles, and that the cipher is extremely non-agile when frequently changing keys. HC is a schizophrenic cipher, performing either extremely well for long streams, or very poorly for short streams. Additionally, the large state size also makes HC unsuitable for many types of hardware implementations.

HC-256 contains two tables, P and Q, each of which contains 1024 32-bit words. The way in which these tables are used is clever when updating P, the g and h functions reference Q as a randomized s-box, and vice versa. These are the components that give the cipher its non-linearity. But the table selection mechanism introduces a degree of overhead unless the cipher is implemented in its restrictive optimization form as described in the HC-256 paper.

HKC modifies HC by abolishing the Q table it maintains a single table consisting of 512 64-bit words, so that its state size is the same as HC-128, but with a 256-bit key.

During the update function g, the table is partitioned in two half tables, where the element being modified is not updated by itself and instead, uses the contents of one element in the other half table. This prevents the cipher from losing entropy.

The f, g and h functions retain the spirit of HC, but with rotation offsets modified to reflect the increase register length. HKC does not distinguish between f_1 and f_2, g_1 and g_2, and h_1 and h_2 tables. In HC, P and Q have separate f, g and h functions, but for mono-tabled HKC, this does not apply.

The initialization phase in HKC is simplified by performing the expansion of the key and IV from 256 bits each to the entire table in-place. Since the diffusion of the f function coupled with element chaining is effective, the third phase of the initialization, which invokes the update function g, only modifies each element of the table once, rather than twice as in HC. This has the effect of further slim-lining the initialization phase.

The designers have not hidden any weaknesses in this cipher.

Associated Data The associated data scheme is similar to that used in Phelix, where the end of the associated data is delineated by combining the length of the data in number of bytes, with the value of one or more of the registers. In HKC, the ideal place to do this is within the MAC register itself.

Authentication The primary difference between HC and HKC is that HKC offers nearly-for-free authentication. There are many ways to do this, and for a time we considered using the main table of HKC as the sink for the plaintext or ciphertext material. However, the diffusion in the cipher is very slow only one element is updated per cycle and for short messages, it would be difficult to randomly select the correct elements that were influenced by message particles. Additionally, it may give the attacker too much control over choosing which elements are emitted into the keystream.

We choose a short external MAC register. During keystream generation and authentication of associated data, the ciphertext is modified and added to the MAC register after each clock of the cipher. This is a simple procedure that takes one word from the main table. The ciphertext is never used to modify the main table, and only indirectly influences the choice of the word taken from the main table. Since the chosen word is assumed to be pseudo-random, the attacker has a difficult time tracking the contents of the MAC (unless he can launch an attack during initialization phase to determine the main table state).

There are sixteen blanking rounds, during which the attacker has no input, to further mix the MAC register. During this stage, the main table restricted to sixteen elements is used as a source of pseudo-random material to add to the MAC register, further confounding the attacker. The modified ciphertext is fed back into the main table during this stage to add to diffusion.

5 Security Analysis

5.1 Randomness of the Keystream of HKC

In this subsection, we investigate the randomness of the keystream of HKC, in a similar way to that for HC-256 [2]. Since HKC uses a 256-bit key, HKC should not be distinguished from an ideal cipher with less than 2^{128} keystream bits, in other words, the goal of the security of HKC against distinguishing attacks should be that there is no distinguishing attack on HKC that requires less than 2^{128} keystream bits. We use the following notation:

- $\dotplus$: addition module 2^{64}
- $\boxplus$: addition module 512
- $\boxminus$: subtraction module 512
- $\langle x \rangle^i$: the i-th least significant bit of an n-bit word x, where $0 \leq i \leq n-1$

First, we have the following result.

Theorem 1 (from [2]). *Let $\mathcal{H}(\cdot)$ be an m-bit-to-n-bit S-box and all the n-bit elements are randomly generated, where $m \geq n$. Let x_1 and x_2 be two m-bit random inputs to $\mathcal{H}(\cdot)$. Then, $\mathcal{H}(x_1) = \mathcal{H}(x_2)$ with probability $2^{-m} + 2^{-n} - 2^{-m-n}$.*

Recall that the feedback function of HKC is

$$\begin{aligned} P[i \bmod 512] &= P[i \bmod 512] \dotplus g(P[i \boxminus 4], P[i \boxplus 1]) \dotplus P[i \boxminus 15] \\ &= P[i \bmod 512] \dotplus P[i \boxminus 15] \dotplus ((P[i \boxminus 4] \ggg 10) \oplus (P[i \boxplus 1] \ggg 35)) \dotplus \\ &\quad P[256 \boxplus ((P[i \boxminus 4] \oplus P[i \boxplus 1]) \bmod 256)], \end{aligned} \tag{1}$$

and the output function of HKC is

$$z_i = h(P[i \boxminus 13]) \oplus P[i \bmod 512].$$

Thus, Eq. (1) can be expressed as

$$\begin{aligned} z_i \oplus h_1(s_i) = (z_{i-512} \oplus h_2(s_{i-512})) \dotplus (z_{i-15} \oplus h_3(s_{i-15})) \dotplus ((z_{i-4} \oplus h_4(s_{i-4})) \ggg 10) \dotplus \\ ((z_{i+1} \oplus h_5(s_{i+1})) \ggg 35) \dotplus P[r_i] \end{aligned} \tag{2}$$

where $h_1(\cdot), h_2(\cdot), h_3(\cdot), h_4(\cdot), h_5(\cdot)$ denote respectively the $h(\cdot)$ function at the $i, i-4, i-15, i-512, i+1$ steps (with respect to the P table),

$$\begin{aligned} s_l &= P[l \boxminus 13] \text{ at the } l\text{-th step, for } l = i, i-4, i-15, i-512, i+1, \\ r_i &= 256 \boxplus ((P[i \boxminus 4] \oplus P[i \boxplus 1]) \bmod 256). \end{aligned}$$

Now, when we focus on the least significant bits of Eq. (2) we have

$$\begin{aligned} \langle z_i \rangle^0 \oplus \langle h_1(s_i) \rangle^0 = \langle z_{i-512} \rangle^0 \oplus \langle h_2(s_{i-512}) \rangle^0 \oplus \langle z_{i-15} \rangle^0 \oplus \langle h_3(s_{i-15}) \rangle^0 \oplus \langle z_{i-4} \rangle^{54} \oplus \\ \langle h_4(s_{i-4}) \rangle^{54} \oplus \langle z_{i+1} \rangle^{29} \oplus \langle h_5(s_{i+1}) \rangle^{29} \oplus \langle P[r_i] \rangle^0. \end{aligned} \tag{3}$$

Further, rewrite Eq. (3) as

$$\begin{aligned} &\langle z_i \rangle^0 \oplus \langle z_{i-512} \rangle^0 \oplus \langle z_{i-15} \rangle^0 \oplus \langle z_{i-4} \rangle^{54} \oplus \langle z_{i+1} \rangle^{29} \\ &= \langle h_1(s_i) \rangle^0 \oplus \langle h_2(s_{i-512}) \rangle^0 \oplus \langle h_3(s_{i-15}) \rangle^0 \oplus \langle h_4(s_{i-4}) \rangle^{54} \oplus \langle h_5(s_{i+1}) \rangle^{29} \oplus \langle P[r_i] \rangle^0. \end{aligned} \tag{4}$$

Assuming the sets of the P elements used as inputs to the h output function for λ consecutive keystream blocks are identical. HKC updates one element of the P table during each step of the keystream generation phase, that means the P tables for generating different keystream blocks are different. This property makes it very hard to devise a distinguishing attack on HKC. Anyway, the sets of the P elements used as inputs to the h output function for different keystream blocks may be identical (i.e. the updated elements of P do not get involved in the generation of the keystream blocks), below we consider a situation that we assume the sets of the P elements used as inputs to the h output function for λ consecutive keystream blocks are identical ($2 \leq \lambda \leq 512$).

Similar to Eq. (4), we can obtain

$$\begin{aligned} &\langle z_j \rangle^0 \oplus \langle z_{j-512} \rangle^0 \oplus \langle z_{j-15} \rangle^0 \oplus \langle z_{j-4} \rangle^{54} \oplus \langle z_{j+1} \rangle^{29} \\ &= \langle h_1(s_j) \rangle^0 \oplus \langle h_2(s_{j-512}) \rangle^0 \oplus \langle h_3(s_{j-15}) \rangle^0 \oplus \langle h_4(s_{j-4}) \rangle^{54} \oplus \langle h_5(s_{j+1}) \rangle^{29} \oplus \langle P[r_j] \rangle^0 \end{aligned} \tag{5}$$

Thus, for the left-hand sides of Eqs. (4) and (5) to be equal, that is

$$\begin{aligned} &\langle z_i \rangle^0 \oplus \langle z_{i-512} \rangle^0 \oplus \langle z_{i-15} \rangle^0 \oplus \langle z_{i-4} \rangle^{54} \oplus \langle z_{i+1} \rangle^{29} \\ &= \langle z_j \rangle^0 \oplus \langle z_{j-512} \rangle^0 \oplus \langle z_{j-15} \rangle^0 \oplus \langle z_{j-4} \rangle^{54} \oplus \langle z_{j+1} \rangle^{29}, \end{aligned} \tag{6}$$

we require that

$$\begin{aligned} &\langle h_1(s_i) \rangle^0 \oplus \langle h_2(s_{i-512}) \rangle^0 \oplus \langle h_3(s_{i-15}) \rangle^0 \oplus \langle h_4(s_{i-4}) \rangle^{54} \oplus \langle h_5(s_{i+1}) \rangle^{29} \oplus \langle P[r_i] \rangle^0 \\ &= \langle h_1(s_j) \rangle^0 \oplus \langle h_2(s_{j-512}) \rangle^0 \oplus \langle h_3(s_{j-15}) \rangle^0 \oplus \langle h_4(s_{j-4}) \rangle^{54} \oplus \langle h_5(s_{j+1}) \rangle^{29} \oplus \langle P[r_j] \rangle^0 \end{aligned} \tag{7}$$

Note that

$$\begin{aligned} s_i &= s_{i-512} \dot{+} g(s_{i-4}, s_{i+1}) \dot{+} s_{i-15}, \\ s_j &= s_{j-512} \dot{+} g(s_{j-4}, s_{j+1}) \dot{+} s_{j-15}. \end{aligned}$$

Approximate Eq. (6) as

$$H(x_1) = H(x_2),$$

where $H(\cdot)$ dentoes a random secret S-box, $x_1 = (\widehat{s}_{i+1}, \widehat{s}_{i-4}, \widehat{s}_{i-15}, \widehat{s}_{i-512}, r_i)$ and $x_2 = (\widehat{s}_{j+1}, \widehat{s}_{j-4}, \widehat{s}_{j-15}, \widehat{s}_{j-512}, r_j)$ are 105-bit random inputs, $\widehat{s}$ denotes bytes (1,4,7) of s.

Since $H(\cdot)$ is a 105-to-1 bit S-box, by Theorem 1 we get that Eq. (7) holds with probability $\frac{1}{2} - 2^{-106}$.

Therefore, for a keystream generated this cipher assuming the sets of the P elements used as inputs to the h function for λ consecutive blocks of keystream are identical, the event described by Eq. (6) happens with probability $\frac{1}{2} - 2^{-106}$. However, for a keystream generated by an ideal cipher, the event described by Eq. (6) happens with probability $\frac{1}{2}$. As a result, it requires 2^{214} many Eq. (6) to distinguish the cipher from an ideal cipher with a success probability 0.9772.

Since the first λ consecutive keystream blocks can make $\binom{\lambda}{2}$ many Eq. (6) and then an additional keystream block will make $\lambda - 1$ many Eq. (6) (together with the previous $\lambda - 1$ keystream blocks), it requires a keystream of $(\frac{2^{214} - \binom{\lambda}{2}}{\lambda - 1} + \lambda) \times 64 \approx \frac{2^{220}}{\lambda - 1}$ bits to

generate 2^{214} many Eq. (6). In particular, in one extreme case $\lambda = 2$, it requires a keystream of about 2^{220} bits to generate 2^{214} many Eq. (6); and in the other extreme case $\lambda = 512$, it requires a keystream of about $\frac{2^{220}}{511} \approx 2^{211}$ bits to generate 2^{214} many Eq. (6).

The probability that the sets of the P elements used as inputs to the h output function for λ consecutive keystream blocks are identical. During the keystream generation phase, only one element of P is updated every step. Clearly, the probability that the sets of the P elements used as inputs to the h output function for the $\lambda = 512$ consecutive keystream blocks are identical is zero.

We below consider $\lambda = 2$ consecutive steps for generating $\lambda = 2$ consecutive keystream blocks z_i and z_{i+1}.

There is only one different element between the P tables used for generating z_i and z_{i+1}, that is the updated element $P[i \boxplus 1]$ before generating z_{i+1}. Recall that $h(x) = P[256 + x_7] \dot{+} P[128 + x_4] \dot{+} P[x_1]$, and $z_{i+1} = h(P[i \boxminus 13]) \oplus P[i \bmod 512]$. Thus, when referring to z_{i+1}, the parameters (x_1, x_4, x_7) correspond to bytes (1,4,7) of $P[i \boxminus 13]$, respectively. Hence, if none of the three parameters $P[256 + x_7], P[128 + x_4]$ and $P[x_1]$ refers to the updated element $P[i \boxplus 1]$, then the sets of the P elements used as inputs to the h output function for the two consecutive keystream blocks z_i and z_{i+1} are identical.

A detailed analysis reveals the following result:

1. When $0 \le i \boxplus 1 \le 127$, only $P[x_1]$ can refer to $P[i \boxplus 1]$.
2. When $128 \le i \boxplus 1 \le 255$, only $P[x_1]$ and $P[128 + x_4]$ can refer to $P[i \boxplus 1]$.
3. When $256 \le i \boxplus 1 \le 383$, only $P[128 + x_4]$ and $P[256 + x_7]$ can refer to $P[i \boxplus 1]$.
4. When $384 \le i \boxplus 1 \le 511$, only $P[256 + x_7]$ can refer to $P[i \boxplus 1]$.

Therefore, on average (for randomly and uniformly distributed i and $P[i \boxminus 13]$), the probability that the sets of the P elements used as inputs to the h output function for the two consecutive keystream blocks z_i and z_{i+1} are identical is

$$\begin{aligned}
&\frac{128}{512} \times (1 - \frac{1}{256}) + \frac{128}{512} \times (1 - \frac{1}{256})^2 + \frac{128}{512} \times (1 - \frac{1}{256})^2 + \frac{128}{512} \times (1 - \frac{1}{256}) \\
&= 1 - \frac{767}{2^{17}} \\
&\approx 0.994.
\end{aligned}$$

Observe that there are five pairs of $h_1(\cdot), h_2(\cdot), h_3(\cdot), h_4(\cdot), h_5(\cdot)$ in Eq. (7), so Eq. (7) holds with a probability of about $0.994^5 \approx 97\%$ when $j = i+1$ (i.e. the case with $\lambda = 2$).

Remarks. From the results from Subsection 2.1 and 2.2, we can see: Even if we assume the event that the sets of the P elements used as inputs to the h output function for the $\lambda = 512$ consecutive keystream blocks are identical held with a one probability, it would require a keystream of about $\frac{2^{214}}{512} \times 64 = 2^{211}$ bits to generate 2^{214} many Eq. (6), which is dramatically larger than the threshold value 2^{128}. Therefore, HKC is secure against distinguishing attacks.

Anyway, an enhancement is to introduce an additional one-byte parameter to the h output function, for example, let $h(x) = P[? + x_7] \dot{+} P[? + x_5] \dot{+} P[? + x_3] \dot{+} P[x_1]$. As a consequence, the resulting Eq. (6) would happen with probability $\frac{1}{2} - 2^{-137}$, and it would require 2^{276} many Eq. (6) to distinguish the cipher from an ideal cipher with a success probability 0.9772.

5.2 Time–Memory Trade-off Attack

HKC takes a 256-bit key and a 256-bit initialization vector (IV), and its internal state is $512 \times 64 = 32768$ bits long. The internal state of HKC is significantly larger than the length specified by the general requirement that the internal state of a stream cipher should be at least as long as the combination of its key and IV. Thus, it is impossible to conduct an efficient time–memory trade-off attack or time–memory–data trade-off attack.

5.3 Correlation and Algebraic Attacks

HKC uses a highly non-linear feedback function $g(\cdot)$ to update the internal state and uses a highly non-linear output function $h(\cdot)$ to generate a keystream block, and each time a keystream block is generated the internal state is updated with $g(\cdot)$. There are 505+512=1017 steps to reach an initial state from the input of a key and an IV. Thus, it is impossible to derive linear combinations that can be used for correlation attacks, or derive a system of sparse equations of low degree for algebraic attacks.

5.4 Guess-and-Determine and Divide-and-Conquer Attacks

After the expansion phase, a word of the internal state, except the first twelve words, depends on every word of the key and IV in a non-linear way. Then, the feedback function in the initialization phase of HKC makes that each word of a key or an IV affects every word of the initial state, and each word of the initial state depends on all the words of the key and IV in a non-linear way. Thus, it is impossible to apply efficient guess-and-determine or divide-and-conquer attacks.

5.5 Related-Key/IV Attack

When expanding a key and an IV to an internal state, HKC uses an update function that involves a step number i in each step. This property makes HKC resist related-key/IV attacks, or slide attacks. After the expansion phase, HKC updates the internal state 512 steps to reach an initial state, using a highly non-linear update function $g(\cdot)$. Thus, it is expected that any difference in the keys/IVs would become indeterminate after the initialization phase.

5.6 MAC Forgery and Differential Plaintext Attack

One of the attacks that is frequently effective on authenticated stream ciphers is that an attacker may replay keyIV pairs with different plaintexts in order to force differences into the state during keystream generation. For HKC, we do not make any guarantees when key-IV pairs are replayed, since the model of usage expressly prohibits this. Nevertheless, during keystream generation, plaintext difference does not propagate into the W table, so does not affect the keystream produced. The key and IV are already significantly mixed by the time they are introduced into the MAC register, so any success the attacker has in revealing the contents of the MAC register are unlikely to lead to recovery of the key. We assume that the attacker may be able to forge some MACs if he replays key-IVs, but this will be difficult due to the significant number of blanking rounds, where the attacker has no input. These blanking rounds provide increased resistance by making use of the non-linear g function in addition to the MAC update routines.

6 Features

6.1 Software

Table 2 shows the throughput of HKC against that of well-known stream cipher RC4, and HC-256, on which HKC is based. The machine on which the cipehrs are implemented is an Intel Core i7 Extreme running at 3.47 GHz. The metric is shown for four packet sizes of 64 bytes, 1 kilobyte, 16 kilobytes, and 1 megabyte respectively. It includes the time taken to initialize the cipher with a key and IV. It is averaged across the time taken to generate one gigabyte of keystream, with refreshing of the key and IV as specified by the packet size.

From the table, we can see that HKC is the fastest of these ciphers. Due to its stream-lined initialization phase, it is approximately four times faster than HC-256 when frequent rekeying occurs, and twenty percent faster for arbitrary length packets, despite the fact that it is additionally computing a MAC. The cipher is faster than RC4 except for the case when it is initialized extremely frequently.

HKC has an advantage in the throughput of software implementation against AES-GCM.

Table 2. Implementation of HKC

Cipher	64 bytes	1 kilobyte	16 kilobytes	1 megabyte
RC4	30.11	16.03	14.85	14.86
HC-256	185.66	15.51	4.97	4.25
HKC	41.83	5.79	3.63	3.54

6.2 Hardware

HKC is primarily a cipher designed for software, as was its predecessor HC-256. This is because it uses a large table, and the majority of its operations are look-ups and modifications of its table elements.

We generated a simulation of a FPGA for Virtex 7 XQ7VX908T, with a clock frequency of 26.147, scaled 64 times to 1.67 GHz. The simulation was generated using Xilinix ISE 14.7.

The area of the FPGA is 34,964 slices, with a throughput of 0.04776 MHz per slice.

7 Intellectual Property

There is no intellectual property about HKC. If any of this information changes, the submitters will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

8 Consent

The submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

9 Conclusion

HKC is a competitive stream cipher that runs at 3.54 cycles/byte on an Intel i7 processor. This is achieved by utilizing 64-bit words rather than 32-bit words, and by parsimony with some features that provide security, given that in almost ten years there have been no effective attacks against HC-256. Unlike HC-256, HKC provides native authentication both for ciphertext, and for unencrypted associated data.

References

1. M. Bellare and C. Namprempre. Authenticated encryption: relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21(4):469–491, 2008.
2. Hongjun Wu. A new stream cipher hc-256. In Bimal K. Roy and Willi Meier, editors, *FSE*, volume 3017 of *Lecture Notes in Computer Science*, pages 226–244. Springer, 2004.

A Recommended Parameter Sets

Length of an initial key: 256 bits
Length of an initialization vector: 256 bits
Length of MAC: 256 bits

B Test Vector

Key: 0x0000000000000000000000000000000000000000000000000000000000000000
IV: 0x0000000000000000000000000000000000000000000000000000000000000000
Message: 0x0000000000000000
Padding: null bytes
Associated data length: 0
Ciphertext : 0xc59f8ada72260723
MAC: 06e8a8763f8a55c8ae1811e0c6e38153306ada08468156af9f89c8c86a75dcc9