Equivalent Keys of HPC

Carl D'Halluin, Gert Bijnens, Bart Preneel*, and Vincent Rijmen**

Katholieke Universiteit Leuven, Dept. Electrical Engineering–ESAT/COSIC
K. Mercierlaan 94, B-3001 Heverlee, Belgium
{carl.dhalluin,gert.bijnens}@esat.kuleuven.ac.be
{bart.preneel,vincent.rijmen}@esat.kuleuven.ac.be

Abstract. This paper presents a weakness in the key schedule of the AES candidate HPC (Hasty Pudding Cipher). It is shown that for the HPC version with a 128-bit key, 1 in 256 keys is weak in the sense that it has 2^{30} equivalent keys. An efficient algorithm is proposed to construct these weak keys and the corresponding equivalent keys. If a weak key is used, it can be recovered by exhaustive search trying only 2^{89} keys on average. This is an improvement by a factor of 2^{38} over a normal exhaustive key search, which requires on average 2^{127} attempts. The weakness also implies that HPC cannot be used in standard constructions for hash functions based on block ciphers. The analysis is extended to HPC with a 192-bit key and a 256-bit key, with similar results. For some other key lengths, all keys are shown to be weak. An example of this is the HPC variant with a 56-bit user key and block length of 128 bits, which can be broken in 2^{31} attempts on average.

1 Introduction

The AES candidate HPC [3] is a block cipher with a variable block length and a variable algorithm: depending on the required block length range, five different versions are defined. In this paper we only look at the version called HPC Medium (sub-cipher number 3), which is the version that supports a block length of 128 bits, as required for the AES candidates. We focus on the HPC expanded key generation. A user key of 128, 192, or 256 bits is expanded to a KX-table containing 256 64-bit words, or 16 384 bits. The ciphertext only depends on the plaintext, the spice[1], and the KX-table. In this paper we assume the spice to be known. Thus if two different user keys K_1 and K_2 generate the same KX-table, then $\texttt{HPC}_{K_1}(P) = \texttt{HPC}_{K_2}(P)$ for every plaintext P. Keys K_1 and K_2 are called *equivalent keys*. In the specifications of HPC [4] it is stated:

> "*Two keys are equivalent if they expand to the same key-expansion table. The likelihood is negligible for keys of size < 1/2 the key-expansion table*

* F.W.O. Research Associate, sponsored by the Fund for Scientific Research - Flanders (Belgium).

** F.W.O. Postdoctoral Researcher, sponsored by the Fund for Scientific Research - Flanders (Belgium).

[1] The *spice* can be regarded as a second key that need not be concealed.

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

size, 8192 bits. For keys longer than this, some will be equivalent, but there is no feasible way to discover an equivalent key pair."

David Wagner announced at the 2nd AES Conference [6] that HPC has equivalent keys based on the generic structure of the key expansion (see Sect. 2.3); he did not analyze their structure. For 128-bit user keys, we have discovered that exactly 2^{-8} of the keys each have 2^{30} equivalent keys. In this paper these keys are called *weak keys*. For 192-bit user keys, approximately 2^{-7} of the keys each have 2^{42} or 2^{32} equivalent keys, and approximately 2^{-16} of the keys each have 2^{74} equivalent keys. For 256-bit user keys, approximately $1.5 \cdot 2^{-7}$ of the keys each have 2^{32} or 2^{34} equivalent keys; approximately 2^{-16} of the keys each have 2^{64} or 2^{66} equivalent keys and approximately 2^{-24} of the keys each have 2^{98} equivalent keys. These results are summarized in Table 1, and are proven in Sections 4 and 6.

Table 1. Approximate number of weak keys and number of equivalent keys

Key length	# weak keys	# equivalent keys
128	2^{120}	2^{30}
192	2^{184}	2^{42}
192	2^{184}	2^{32}
192	2^{176}	2^{72}
256	2^{249}	2^{32}
256	2^{248}	2^{34}
256	2^{241}	2^{66}
256	2^{240}	2^{64}
256	2^{232}	2^{98}

In Sect. 2 we explain how the expanded key table is calculated. Section 3 clarifies how the key expansion results in equivalent keys. In Sect. 4 we compute the number of weak keys and we present a method to construct weak keys. The impact of weak keys on exhaustive key search is treated in Sect. 5. In Sect. 6 we briefly search for weak keys if a user key with length other than 128 bits is used. In Sect. 7 we show that HPC-based hash functions are insecure, and in Sect. 8 we discuss how the weak keys can be avoided. Finally we conclude in Sect. 9.

In this paper all numbers in hexadecimal notation (indicated with a subscript x) are written with the least significant byte on the right side. Numbers in decimal notation have no subscript or a subscript d. The least significant bit (that is, the rightmost bit) is numbered bit 0.

2 HPC Expanded Key Generation

The expanded key table (denoted by `KX`-table) contains 286 64-bit words. The `KX`-table depends on the user key and on the sub-cipher number `sc`. The last 30

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

entries of the KX-table are equal to the first 30:

$$\texttt{KX[i+256]} = \texttt{KX[i]} \quad \text{for } i = 0, 1, \ldots, 29.$$

This means that the KX-table effectively contains $256 \cdot 64 = 16\,384$ bits.

The KX-table is calculated in four steps. Firstly the table is filled with 256 pseudo-random values. Secondly the user key is XORed into the table. The goal of the *stirring function* is to make all the 256 entries of the table depend on the user key. Finally the last 30 entries of the table are set equal to the first 30. We discuss these steps in more detail below.

2.1 Filling the KX-table with Pseudo-random Values

The first entries of the KX-table are initialized using three mathematical constants (with sc denoting the sub-cipher number):

$$\texttt{KX[0]} = \texttt{PI19 + sc} \tag{1}$$

$$\texttt{KX[1]} = \texttt{E19 * the key length} \tag{2}$$

$$\texttt{KX[2]} = \texttt{R220 rotated left over sc bits} \tag{3}$$

where $\texttt{PI19} = 3141592653589793238_d$, $\texttt{E19} = 2718281828459045235_d$, $\texttt{R220} = 1414213562373095048 8_d$, $\texttt{sc} = 3$, and the key length is 128, 192, or 256.

Now the remaining 253 words of the table are pseudo-randomly filled with the following equation for $i = 3, \ldots, 255$, where '^' denotes the XOR operation and '>>', '<<' denote right and left shift operations respectively.

```
KX[i] = KX[i-1] + (KX[i-2] ^ KX[i-3]>>23 ^ KX[i-3]<<41) .    (4)
```

2.2 XORing the User Key into the KX-table

The user key is XORed into the first entries of the KX-table. If the user key K has 128 bits, we have $\texttt{K} = \texttt{K}_\texttt{H} \parallel \texttt{K}_\texttt{L}$ with $\texttt{K}_\texttt{H}$ the most significant 64 bits of K, and $\texttt{K}_\texttt{L}$ the least significant 64 bits of K, and $\parallel$ the concatenation symbol. We have (using C notation)

$$\texttt{KX[0]} \texttt{\^{}=} \texttt{K}_\texttt{L}$$
$$\texttt{KX[1]} \texttt{\^{}=} \texttt{K}_\texttt{H} \ .$$

If the length of the user key is 192 bits or 256 bits, then we have to XOR the appropriate part of the key to KX[2] and KX[3] as well.

2.3 Stirring Function

The whole KX-table is made key dependent by means of the iterative stirring function. This function has eight internal state variables s0 to s7, that are initialized with s0 = KX[248], ..., s7 = KX[255]. The stirring function is run three times

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

Fig. 1. Pseudo-code (C notation) for the stirring function (the numbers are written in decimal notation)

```
for (j=0; j<3; j++)       /* Number of passes is 3        */
  for (i=0; i<256; i++) { /* Run over the entire KX-table */
 (1)    s0 ^= (KX[i] ^ KX[(i+83)&255]) + KX[s0&255]
 (2)    s1 += s0
 (3)    s3 ^= s2
 (4)    s5 -= s4
 (5)    s7 ^= s6
 (6)    s3 += s0>>13
 (7)    s4 ^= s1<<11
 (8)    s5 ^= s3<<(s1&31)
 (9)    s6 += s2>>17
(10)    s7 |= s3+s4
(11)    s2 -= s5
(12)    s0 -= s6^i
(13)    s1 ^= s5 + PI19
(14)    s2 += s7>>j
(15)    s2 ^= s1
(16)    s4 -= s3
(17)    s6 ^= s5
(18)    s0 += s7
(19)    KX[i] = s2 + s6
        }
```

(j-loop) over the entire KX-table (i-loop), allowing each bit to influence every other bit. The code for the stirring function is given in Fig. 1.

We denote the set of eight state variables before pass i, j of the stirring function by $\mathbf{s_{i,j}}(K)$ where K is the user key. We denote the stirring function of pass i, j by $F_{i,j,K}$. Hence the initial internal state is $\mathbf{s_{0,0}}(K)$. Note that $\mathbf{s_{0,0}}(K)$ does not depend on the user key K. The internal state after the stirring function is denoted by $\mathbf{s_{final}}(K)$. Thus we have

$$\begin{aligned}
\mathbf{s_{final}}(K) &= F_{255,2,K}(\mathbf{s_{255,2}}(K)) \\
&= F_{255,2,K}(F_{254,2,K}(\mathbf{s_{254,2}}(K))) \\
&= F_{255,2,K}(F_{254,2,K}(\cdots(F_{0,2,K}(\mathbf{s_{0,2}}(K))))) \\
\mathbf{s_{0,2}}(K) &= F_{255,1,K}(\mathbf{s_{255,1}}(K)) \ .
\end{aligned}$$

This leads to the following iterated definition:

$$\begin{aligned}
\mathbf{s_{i,j}}(K) &= F_{i-1,j,K}(\mathbf{s_{i-1,j}}(K)) \text{ for } i = 1, 2, \ldots, 255 \text{ and } j = 0, 1, 2, \\
\mathbf{s_{0,j}}(K) &= F_{255,j-1,K}(\mathbf{s_{255,j-1}}(K)) \text{ for } j = 1, 2.
\end{aligned}$$

If we consider the stirring function for the HPC variant with 128-bit user key, we note that only $F_{i,j,K}$, where $0 \le i \le 1$ and $0 \le j \le 2$, depend on the user key. Thus if $\mathbf{s_{i_0,j_0}}(K_1) = \mathbf{s_{i_0,j_0}}(K_2)$ for two different user keys K_1 and K_2 and

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

$i_0 \geq 1$, we know that $\mathbf{s_{i,j_0}}(K_1) = \mathbf{s_{i,j_0}}(K_2)$ for $i_0 \leq i \leq 255$, and we also know that $\mathbf{s_{0,j_0+1}}(K_1) = \mathbf{s_{0,j_0+1}}(K_2)$ if $j_0 = 0, 1$ or that $\mathbf{s_{final}}(K_1) = \mathbf{s_{final}}(K_2)$ if $j_0 = 2$. We can even control $F_{0,j,K}$ and $F_{1,j,K}$ independently, since $F_{0,j,K}$ only depends on the least significant half of the user key, and $F_{1,j,K}$ only depends on the most significant half.

These considerations show that if we can find a set of user keys $\{K_a\}$ such that $F_{1,j,K_a}(\mathbf{s_{1,j}}(K_a))$ and $F_{0,j,K_a}(\mathbf{s_{0,j}}(K_a))$ are constant over the entire set $\{K_a\}$ and only depend on j, then the `KX`-table will also be constant for all user keys in the set $\{K_a\}$. This means that we can define an *equivalence relationship* in the set of all user keys; the condition for equivalence is : "expands into the same `KX`-table as." The disjunct set $\{K_a\}$ is called an *equivalence class*.

In this paper we will show that such equivalence classes do exist. For the 128-bit variant of HPC, we can find 2^{90} equivalence classes, each containing 2^{30} elements.

3 Equivalent Keys (Key Length Equal to 128 Bits)

Before the stirring function is applied to the `KX`-table, the user key is XORed into the `KX`-table (see Sect. 2.2). For the AES candidate HPC with user key length equal to 128 bits, only `KX[0]` and `KX[1]` are influenced by the user key. The other values `KX[2]` up to `KX[255]` are independent of the 128-bit user key. Of course this is only true *before* the stirring function is applied to the `KX`-table.

A closer investigation of the key expansion leads us to an equation, from now on called the *dangerous equation*:

```
s0 ^= (KX[i] ^ KX[(i+83)&255]) + KX[s0&255] .        (5)
```

This equation is dangerous because when `s0` & $255_d = i$, different values of `KX[i]` can produce the same result. We investigate this problem closer for $i = 0$ and $i = 1$.

3.1 Dangerous Equation for $i = 0$

First we check the dangerous equation (5) for $i = 0$. The initial values of `s0` to `s7` are constants for key length 128 bits and sub-cipher number 3, and are equal to the initial values of respectively `KX[248]` to `KX[255]`. Their values are shown in Table 2. The initial value of `KX[0]` is equal to (`PI19`+`sub-cipher number`) ^ $\mathtt{K_L}$. Thus we have

$$\mathtt{KX[0]} = \mathtt{2b992ddf\ a23249d9}_x \ \hat{}\ \mathtt{K_L} \ .$$

Since `s0` & $255_d = \mathtt{4b}_x = 75_d$, the dangerous equation for $i = 0$ becomes:

```
s0 ^= (KX[0] ^ KX[83]) + KX[75]  .
```

The values `KX[83]` and `KX[75]` are constants (for key length equal to 128 and sub-cipher number equal to 3) and are equal to

$$\mathtt{KX[83]} = \mathtt{093817dc\ b93586e6}_x \ ,$$
$$\mathtt{KX[75]} = \mathtt{989a3714\ d85dee74}_x \ .$$

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

If we evaluate the dangerous equation numerically, we find

$$\texttt{s0 ^= (22a13a03 1b07cf3f}_x \texttt{ ^ K}_\texttt{L}\texttt{) + 989a3714 d85dee74}_x \ .$$

This mapping from $\texttt{K}_\texttt{L}$ to s0 is clearly a bijection, hence a modification of $\texttt{K}_\texttt{L}$ induces a modification of s0. The state variables s0 to s7 all change with high probability due to the stirring function (see Sect. 2.3) and the whole KX-table changes significantly. Hence we cannot find equivalent keys that differ in the least significant 64 bits of the user key.

Table 2. Initial state for the stirring function of HPC with user key length equal to 128 bits (all values in hexadecimal notation)

s0	$\texttt{4cfd66f0 5ab4064b}_x$
s1	$\texttt{6f7d4e0e 4107bd8c}_x$
s2	$\texttt{eadadb90 0f4b3d2a}_x$
s3	$\texttt{f24cb427 cb159a63}_x$
s4	$\texttt{d7ee7776 c0ecbc0b}_x$
s5	$\texttt{3c255969 3f8f7688}_x$
s6	$\texttt{390009fb 99146a25}_x$
s7	$\texttt{1e5d58c2 7c76052e}_x$

3.2 Dangerous Equation for $i = 1$

If $\texttt{K}_\texttt{L}$ is constant, then the state variables after the first pass of the i-loop of the stirring function ($i = 0$) are constant as well. For $i = 1$, the initial state of s0 is part of the internal state of the stirring function, after one pass of the i-loop ($i = 0$). For now we will assume that we can choose KX[0] such that the least significant byte of s0 after the first pass of the i-loop ($i = 0$) of the stirring function is equal to $\texttt{01}_x$ (see Sect. 4). The dangerous equation (5) for the case $i = 1$ becomes:

$$\texttt{s0 ^= (KX[1] ^ KX[84]) + KX[1]} \ .$$

Let T denote (KX[1] ^ KX[84]) + KX[1]. The value KX[84] is completely determined by the key length (128) and the sub-cipher number (3). We have

$$\texttt{KX[84]} = \texttt{a8f07353 9f208716}_x \ .$$

If bit ω' of KX[84] is set to 1, then it does not matter whether bit ω' of KX[1] is set to 0 or 1. This can easily be shown by looking at the individual bits of T. The bit on position t is denoted by a subscript t, with $t = 0$ for the least significant bit. We obtain for $0 \le t \le 63$:

$$\begin{aligned}
\texttt{T}_t &= \texttt{KX[1]}_t \texttt{ ^ KX[84]}_t \texttt{ ^ KX[1]}_t \texttt{ ^ } c_t \\
&= \texttt{KX[84]}_t \texttt{ ^ } c_t \ , \\
c_{t+1} &= (\texttt{KX[1]}_t \texttt{ ^ KX[84]}_t)\texttt{KX[1]}_t \texttt{ ^ } (\texttt{KX[1]}_t \texttt{ ^ KX[84]}_t \texttt{ ^ KX[1]}_t)c_t \\
&= (\texttt{KX[1]}_t \texttt{ ^ KX[84]}_t)\texttt{KX[1]}_t \texttt{ ^ KX[84]}_t c_t \ ,
\end{aligned}$$

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

with c_{t+1} the carry bit generated at bit position t and $c_0 = \texttt{0}$. Investigation of these equations leads us to three observations:

1. $\texttt{T}_t$ does not depend on $\texttt{KX[1]}_t$;
2. c_{t+1} does not depend on $\texttt{KX[1]}_t$ if and only if $\texttt{KX[84]}_t = \texttt{1}$;
3. c_{t+1} is not used for $t = 63$, since we work with 64-bit quantities.

The support of KX[84] (set of bit positions on which we have a 1), is denoted by Ω'. Examining the numerical value of KX[84], we find that $\Omega' = supp(\texttt{KX[84]}) = \{$ 1, 2, 4, 8, 9, 10, 15, 21, 24, 25, 26, 27, 28, 31, 32, 33, 36, 38, 40, 41, 44, 45, 46, 52, 53, 54, 55, 59, 61, 63 $\}$. We can thus complement the value of any combination of bits on positions $\omega' \in \Omega' \cup \{63\}$ of KX[1] without changing the value of T = (KX[1]^KX[84])+KX[1]. Since the set $\Omega' \cup \{63\}$ contains 30 elements, we can choose 30 bits of KX[1] at random, without changing the value of the right half of the dangerous equation. As $\texttt{KX[1]} = \texttt{dca375e0 59b0b980}_x \ \hat{}\ \texttt{K}_\texttt{H}$, we can easily find a set of 2^{30} equivalent keys, by complementing the bit positions ω' of $\texttt{K}_\texttt{H}$.

Now we have shown that if we can set the least significant byte of s0 equal to $\texttt{01}_x$ (after one pass of the i-loop ($i = 0$) of the stirring function), then we can construct 2^{30} equivalent keys by simply complementing the bits on a subset of 30 specific bit positions ω' of $\texttt{K}_\texttt{H}$. In the next section it is shown that the least significant byte of s0 can be set equal to $\texttt{01}_x$ by choosing specifically bits 0 to 7 and bits 13 to 20 of $\texttt{K}_\texttt{L}$.

If we look at a 128-bit weak key, then the set of bit positions ω that can be complemented, is equal to $\Omega = \{\omega \mid \omega = \omega' + 64, \text{ with } \omega' \in \Omega'\} \cup \{127\}$.

4 Counting Weak Keys and a Construction Method

We start by analyzing the first pass of the i-loop of the stirring function ($i = 0$). We use a specific notation to calculate the value of the least significant byte of s0 after the 19 steps: $\texttt{s0}_n$ indicates the value of s0 before step n. $\texttt{s0}_1$ is the initial value of s0 (before the stirring function is applied). $\texttt{s0}_{\text{end}}$ is the value of s0 after the first pass of the stirring function for $i = 0$. In this calculation we only have to keep track of the least significant byte of s0. If a certain operation does not affect this byte, the operation is not taken into account (e.g., addition with $\texttt{s1}_7 \ll 11$). We substitute the values of Table 2 in the equations. We have:

$$\texttt{s0}_{\text{end}} = \texttt{s0}_{18} + \texttt{s7}_{18} \ ,$$

which can be reworked (see Fig. 1) to:

$$\texttt{s0}_{\text{end}} = \texttt{s0}_2 + \texttt{36}_x + (\texttt{0b}_x | (\texttt{54}_x + \texttt{s0}_2 \gg 13)) \ . \tag{6}$$

As shown in Sect. 3, we have a weak key if the least significant byte of $\texttt{s0}_{\text{end}}$ is equal to $\texttt{01}_x$. If we take a look at (6) we see the OR operation (denoted by '|') with the fixed value $\texttt{0b}_x$, which forces the least significant 4 bits of the result to be either $\texttt{b}_x$ or $\texttt{f}_x$. If we want $\texttt{s0}_{\text{end}}$ to be $\texttt{01}_x$, we must put restrictions on

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

the least significant 4 bits of $\mathtt{s0}_2 + \mathtt{36}_x$. This can be translated to the following condition on K_L:

$$K_L \bmod 10_x = 8 \text{ or } K_L \bmod 10_x = c_x \ .$$

Appendix A provides further details on the construction of the weak keys. Some examples are given in Table 3.

Table 3. Examples of weak keys of the 128-bit variant of HPC (all values in hexadecimal notation)

00000000 00000000 00000000 00004008$_x$
008ecc3c 9299f88d bc08b82f edbcb3ec$_x$
f0eabdac 5446cfa4 165f3da0 b829d24c$_x$
9afc9c44 f5a2b0eb 47943add a82388bc$_x$
a20c6bb9 079f1a27 8d54f3db f5e30828$_x$
21c7d6c0 2111eec9 c4def468 cbe626d8$_x$
a655b17c 7fc18ed8 a8bc9e70 ff7de5ec$_x$
e6d6685d d6f12b9d 7cc71ae6 d23584ac$_x$
ccaea4eb b12fae2e 22b4393b e33cfbec$_x$
5195c9f0 5da0798c 86b3a27c 2c1ae97c$_x$
d90ad927 d087f410 41bdea99 3c725338$_x$

5 Exhaustive Key Search for User Key Length 128 Bits

If we know that someone uses a weak key, we can find that key by exhaustive key search, trying on average 2^{89} different key values. We only have to construct every weak key using the procedure given in Appendix A and check whether it is the correct key. In this procedure we can choose freely 120 bits of the user key. Since the values of 30 bits of $\mathtt{K_H}$ do not influence the KX-array, we can assign arbitrary values to these bits. Hence only $120 - 30 = 90$ key bits remain to be recovered. This is a major improvement compared to a brute force attack in which we have to recover 128 bits (or 120 bits if we know that we have a weak key).

Even if we do not know whether a key is weak, an exhaustive key search can be improved by starting the search with weak keys. In 1 case out of 256 the user key will be weak, which implies that the search will be successful after at most 2^{90} encryptions.

6 HPC with Other Key Lengths

The key length influences the KX-table due through the initial value of KX[1], see (2), and the recursion formula (4). We briefly study the AES candidate HPC with key length equal to 192 and 256 bits. The results are summarized in Table 1. Finally we take a short look at some other key lengths.

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

6.1 Key Length 192 Bits

Since the number of key words increases from two to three, we evaluate the dangerous equation (5) three times:

$$i = 0: \texttt{s0 ^= (KX[0] ^ KX[83]) + KX[s0\&255]} \quad (7)$$

$$i = 1: \texttt{s0 ^= (KX[1] ^ KX[84]) + KX[s0\&255]} \quad (8)$$

$$i = 2: \texttt{s0 ^= (KX[2] ^ KX[85]) + KX[s0\&255]}\ . \quad (9)$$

As before, it turns out that initially, $\texttt{s0}\&255 \neq 0$, hence there are no weak keys for $i = 0$. In the cases $i = 1$ and $i = 2$ however, `s0` depends on some key bits (see Sect. 4). This allows us to generate two sets of 2^{184} weak keys (together a fraction of approximately $2^{-7} - 2^{-16}$ of the key space) with a different number of equivalent keys. These two sets have an intersection containing approximately 2^{172} weak keys ($\approx 2^{-16}$ of the key space) with 2^{74} equivalent keys each.

- The case $i = 1$ is the same as for 128-bit keys. Hence, we know that 1 key in 256 is weak. We also know that the number of equivalent keys is determined by the Hamming weight of `KX[84]`, and the most significant bit of `KX[84]`. Calculations lead us to $\texttt{KX[84]} = \texttt{efc64dbb cb9f7b71}_x$ with Hamming weight 42. The most significant bit is set to `1`. This means that each weak key has 2^{42} equivalent keys. Two equivalent keys in this class differ only in the middle 64 bits. If one knows in advance that someone uses a weak key, the key is found by exhaustive search after on average 2^{141} attempts.
- For the case $i = 2$ we also know that 1 key in 256 is weak. Since $\texttt{KX[85]} = \texttt{8842f3d7 13b09bab}_x$ with Hamming weight 32 and most significant bit equal to `1`, each weak key has 2^{32} equivalent keys. Two equivalent keys differ only in the most significant 64 bits.
- Simulations show that we can combine the two previous cases[2]. We find that approximately 2^{-16} of the user keys are weak, corresponding to 2^{74} equivalent keys each. Two equivalent keys can differ in the most significant 128 bits.

6.2 Key Length 256 Bits

For key length equal to 256 bits, we have to evaluate the dangerous equation (5) for $i = 1$, $i = 2$, and $i = 3$. Now we can generate three sets of 2^{248} weak keys with a different number of equivalent keys. The intersections between these three sets are non-empty and contain weak keys with a significant number of equivalent keys.

- For the case $i = 1$ we have $\texttt{KX[84]} = \texttt{e8e10c4d d1eb6c1d}_x$ with Hamming weight 32 and the most significant bit is set to `1`. Hence 1 in 256 user keys is weak, with 2^{32} equivalent keys.

[2] There is no reason why these two events should be independent. Computer simulations show however that this assumption results in a reasonable approximation.

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

- For the case $i = 2$ we have $\texttt{KX[85]} = \texttt{6cd2af08 790165f3}_x$ with Hamming weight 31 but the most significant bit is set to 0. Hence 1 in 256 user keys is weak, with 2^{32} equivalent keys.
- For the case $i = 3$ we have $\texttt{KX[86]} = \texttt{ec7833a4 9d9bce38}_x$ with Hamming weight 34 and the most significant bit is set to 1. Hence 1 in 256 user keys is weak, with 2^{34} equivalent keys.
- We can combine the previous three cases, and generate four other sets of weak keys with a different number of equivalent keys. An overview is given in Table 1. There are approximately An example of a weak key with approximately 2^{232} weak keys with 2^{98} equivalent keys; an example of such a key is $\texttt{K} = [\texttt{K}_3 \parallel \texttt{K}_2 \parallel \texttt{K}_1 \parallel \texttt{K}_0]$, where $\texttt{K}_i$ denote 64-bit quantities, and

$$[\texttt{K}_3 \parallel \texttt{K}_2] = \texttt{aabe6c6d e3a06a02 b7650b34 6c73cf94}_x \ , \tag{10}$$

$$[\texttt{K}_1 \parallel \texttt{K}_0] = \texttt{2900e495 27f8eaba a44524e0 bca9cd27}_x \ . \tag{11}$$

6.3 Other Key Lengths

If we keep the sub-cipher number constant, the entries of the KX-array before the application of the stirring function depend on the key length only (see equations (2) and (4)). If we can find key lengths such that $\texttt{KX[248]} \ \& \ 255_d = 00_x$, then evaluating the dangerous equation (5) for $i = 0$ gives

```
s0 ^= (KX[0] ^ KX[83]) + KX[0] ,
```

which means that *all keys* are weak keys. Table 4 shows the first 10 key lengths for which all the keys are weak. Their number of equivalent keys is also shown. It is an interesting coincidence that for a key length of 56 bits (as for the DES [1]), all keys are weak keys with 2^{24} equivalent keys. This means that the user key can be recovered by exhaustive key search, after on average 2^{31} attempts. For the sake of clarity we repeat that we are studying the HPC version with 128-bit block length. The version with 64-bit block lengths, as the DES, has a different sub-cipher number and has no weak keys.

Table 4. Key lengths for which all the keys are weak, and their number of equivalent keys

Key length	KX[83]	# Equivalent keys
56	$\texttt{87d0e495 0b884dc5}_x$	2^{24}
403	$\texttt{387d8891 8dbf8aa7}_x$	2^{34}
608	$\texttt{3ff84d03 8d713835}_x$	2^{33}
1190	$\texttt{d5d15fd2 86e68311}_x$	2^{32}
1993	$\texttt{3d16d709 7971336a}_x$	2^{34}
2491	$\texttt{1029f33c e3daa470}_x$	2^{31}
2512	$\texttt{64b16068 745df8ea}_x$	2^{32}
2656	$\texttt{593f5ca5 e7b6a6ad}_x$	2^{39}
2983	$\texttt{fa06ddd3 f052af40}_x$	2^{33}
3245	$\texttt{f28fb98c 7b26832f}_x$	2^{35}

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

7 HPC-Based Hash Functions

The existence of such a large number of equivalent keys has a serious impact on the use of HPC in the standard constructions for hash functions based on block ciphers. The problems of weak keys in this context have already been discussed earlier, see for example [2].

The building block of most hash functions based on block ciphers is known as the Davies-Meyer hash function (although the authorship of this function is uncertain). This is an iterated hash function; in step i, the chaining variable H_{i-1} and the ith message block X_i are compressed to the next value of the chaining variable H_i as follows:

$$H_i = E_{X_i}(H_{i-1}) \hat{} H_{i-1} \ ,$$

where $E_K()$ denotes encryption with a block cipher E using the key K. This mapping is denoted the *compression function*.

A first observation is that equivalent keys for the block cipher E lead to trivial collisions for the hash function. Indeed, if the X_i's are chosen from a single equivalence class (that contains 2^{30} keys), the value of H_i will not change. A second observation is that equivalent keys lead to trivial 2nd preimage attacks in a similar way. The fact that in 1 case out of 256 key search can be sped up with a factor of 2^{38} implies that finding a 2nd preimage can take advantage of the same speed-up.

By applying an affine transformation of variables (for example, swapping X_i and H_{i-1}), the above attacks may become attacks on the compression function rather than on the hash function. Nevertheless, a strong compression function is a desirable criterion (see [2] for more details on the relation between weak keys and the strength of the compression function).

8 How to Eliminate the Weak Keys

R. Schroeppel has announced a 'tweak' of HPC that should eliminate the weak keys [5]. The following line of code is added to the stirring function after line (1) (see Fig. 1).

```
(1a)   s2 += KX[i]
```

This ensures indeed that changes in the user key are propagated to the entire state, which implies that with very high probability no two 'short' user keys will result in the same `KX` table. An alternative solution would be to create part of the `KX` table as a bijection of the user key. This would guarantee that no two user keys are equivalent.

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

9 Conclusion

In this paper we have discussed a serious weakness of the key expansion of AES candidate HPC. For the 128-bit user key version, it is shown that exactly 2^{-8} of the user keys are weak keys, each with 2^{30} equivalent keys. These weak keys and their corresponding equivalent keys can be constructed using a very simple and efficient algorithm. Such a weak key can be found by exhaustive key search, trying only 2^{90} different keys.

We investigated the presence of weak keys for user key length 192 and 256 bits, and showed that respectively 2^{-7} and $1.5 \cdot 2^{-7}$ of the user keys are weak keys. We even found key lengths, for which all user keys are weak. Finally we note that HPC in its present form is not suitable for use in hash function constructions.

Acknowledgment The authors would like to thank R. Schroeppel for motivating their research by announcing an attractive prize for the best cryptanalysis of HPC.

References

1. FIPS 46, "Data encryption standard," NBS, U.S. Department of Commerce, Washington D.C., Jan. 1977.
2. B. Preneel, R. Govaerts, J. Vandewalle, "Hash functions based on block ciphers: a synthetic approach," *Advances in Cryptology, Proceedings Crypto'93, LNCS 773*, D. Stinson, Ed., Springer-Verlag, 1994, pp. 368–378.
3. R. Schroeppel, "An overview of the Hasty Pudding Cipher," AES-submission, `http://www.cs.arizona.edu/~rcs/hpc`, 1998.
4. R. Schroeppel, "The Hasty Pudding Cipher: Specific NIST requirements," AES-submission, 1998.
5. R. Schroeppel, "Tweaking the Hasty Pudding Cipher," `http://www.cs.arizona.edu/~rcs/hpc/tweak`, 1999.
6. D. Wagner, "Equivalent keys for HPC," rump session talk at the 2nd AES Conference, Rome (I), March 22-23, 1999, `http://www.cs.berkeley.edu/~daw`

A Construction of 128-bit Weak Keys

In this appendix we show how to construct the weak 128-bit user keys. The construction of 192-bit and 256-bit weak keys is omitted due to space restrictions.

To construct a 128-bit weak key we have to satisfy (6) for the least significant byte of $\mathtt{s0}_2$:

$$\mathtt{01}_x = \mathtt{s0}_2 + \mathtt{36}_x + \mathtt{0b}_x \,|\, (\mathtt{54}_x + \mathtt{s0}_2 \gg 13) \tag{12}$$

$$\Updownarrow$$

$$\mathtt{L} = \mathtt{0b}_x | \mathtt{R} \ , \tag{13}$$

where $\mathtt{L} = \mathtt{cb}_x - \mathtt{s0}_2$ and $\mathtt{R} = (\mathtt{54}_x + \mathtt{s0}_2 \gg 13)$. In order to satisfy this equation, bits 0, 1, and 3 of $\mathtt{L}$ must be $\mathtt{1}$. In Sect. 4 it is proven that in order to set these

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

bits to 1, we have to choose $\mathtt{K_L}$ & $\mathtt{f}_x$ equal to $\mathtt{8}_x$ or $\mathtt{c}_x$. A weak key can be constructed as follows:

- Assign a random value to bits 2, 4, 5, 6 and 7 of $\mathtt{K_L}$. Set bits 0 and 1 of $\mathtt{K_L}$ equal to 0, and set bit 3 of $\mathtt{K_L}$ equal to 1.
- Calculate the least significant byte of $\mathtt{s0}_2 = \mathtt{s0}_1\mathtt{\hat{}}\Big(\mathtt{(KX[0]\ \hat{}\ KX[83])} + \mathtt{KX[75]}\Big) = \mathtt{4b}_x\mathtt{\hat{}}\Big((\mathtt{d9}_x\mathtt{\hat{}}\mathtt{K_L}\mathtt{\hat{}}\mathtt{e6}_x) + \mathtt{74}_x\Big)$.
- Calculate the byte value $\mathtt{L} = \mathtt{cb}_x - \mathtt{s0}_2 = \mathtt{0b}_x|\mathtt{R}$. Then calculate R

$$\begin{aligned}\mathtt{R} &= \mathtt{54}_x + \left(\mathtt{s0}_1\mathtt{\hat{}}\Big((\mathtt{KX[0]\hat{}KX[83]}) + \mathtt{KX[75]}\Big)\right) \gg 13 \\ &= \mathtt{54}_x + \left(\Big(\mathtt{s0}_1 \gg 13\Big)\mathtt{\hat{}}\Big((\mathtt{KX[0]\hat{}KX[83]}) \gg 13 + \mathtt{KX[75]} \gg 13 + p\Big)\right) \\ &= \mathtt{54}_x + \left(\mathtt{a0}_x\mathtt{\hat{}}\left(\Big(\mathtt{92}_x\mathtt{\hat{}}(\mathtt{K_L} \gg 13)\mathtt{\hat{}}\mathtt{ac}_x\Big) + \mathtt{ef}_x + p\right)\right)\ ,\end{aligned}$$

  where p is a carry bit, which we have to calculate first.
- To calculate p, choose at random bits 8 to 12 of $\mathtt{K_L}$ and calculate the carry bit p of `((KX[0] ^ KX[83]) + KX[75])`. The bit $p = 1$ if and only if the value of this expression, calculated only with the 13 least significant bits of KX[0], KX[83] and KX[75], is larger than $2^{13} - 1$. In the other case $p = 0$. Hence the choice of bits 0 to 12 of $\mathtt{K_L}$ determines the value of p.
- Now we have to solve the equation $\mathtt{L} = \mathtt{0b}_x|\mathtt{R}$. First assign a random value to bits 0, 1 and 3 of R. The unknown values in this equation are bits 13 to 20 of $\mathtt{K_L}$. Now we can solve for $\mathtt{K_L} \gg 13$, and we find exactly one solution.
- Now we have a value for bits 0 to 20 of $\mathtt{K_L}$. These 21 bits are completely determined by choosing 13 random bits (bits 2, 4, 5, 6, 7 and 8 to 12 of $\mathtt{K_L}$, and bits 0, 1 and 3 of R).
- We can choose the other $64 - 21 = 43$ bits of $\mathtt{K_L}$ and the 64 bits of $\mathtt{K_H}$ at random. We now have obtained a weak key. This means that the number of weak keys is exactly 2^{120} (since we chose freely 120 bits to construct the weak key). This means that exactly 2^{-8} of the keys are weak keys. In Sect. 3 it is proven that each weak key has 2^{30} equivalent keys.

Example

We now construct a weak key, using the techniques described in the previous section.

- We assign the bit value 0 to bits 2, 4, 5, 6 and 7 of $\mathtt{K_L}$. We set bit 0 and 1 of $\mathtt{K_L}$ equal to 0, and set bit 3 of $\mathtt{K_L}$ equal to 1. Then the least significant byte of the key has the value $\mathtt{08}_x$.
- We calculate the least significant byte of $\mathtt{s0}_2$. This gives $\mathtt{e0}_x$.
- We set bits 8 to 12 of $\mathtt{K_L}$ to 0. It turns out that $p = 1$.

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.

- We calculate $\mathtt{L} = \mathtt{cb}_x - \mathtt{s0}_2 = \mathtt{eb}_x$. We set bits 0, 1 and 3 of $\mathtt{R}$ to $\mathtt{0}$. We solve the equation $\mathtt{R} = \mathtt{e0}_x$. It turns out that $(\mathtt{K_L} \gg 13) = \mathtt{02}_x$.
- The other bits 21–63 of $\mathtt{K_L}$ and 0–63 of $\mathtt{K_H}$ are set $\mathtt{0}$. Now we have constructed the weak key

$$\mathtt{K} = [\mathtt{K_H} \parallel \mathtt{K_L}] = \mathtt{00000000\ 00000000\ 00000000\ 00004008}_x \ .$$

It is easy to check that this key is indeed a weak key with 2^{30} equivalent keys.

Appeared in *Advances in Cryptology – ASIACRYPT 1999*, Lecture Notes in Computer Science 1716, K. Lam, E. Okamoto, and C. Xing (eds.), Springer-Verlag, pp. 29–42, 1999.