Related-key Attacks Against Full Hummingbird-2

Markku-Juhani O. Saarinen ⋆

CBEAM
mjos@iki.fi

Abstract. We present attacks on full Hummingbird-2 which are able to recover the 128-bit secret keys of two black box cipher instances that have a certain type of low-weight XOR difference in their keys. We call these highly correlated keys as they produce the same ciphertext with a significant probability. The complexity of our main chosen-IV key-recovery attack is 2^{64}. The first 64 bits of the key can be independently recovered with only 2^{36} effort. This is the first sub-exhaustive attack on the full cipher under two related keys. Our attacks use some novel tricks and techniques which are made possible by Hummingbird-2's unique word-based structure. We have verified the correctness and complexity of our attacks by fully implementing them. We also discuss enabling factors of these attacks and describe an alternative design for the WD16 nonlinear keyed function which is resistant to attacks of this type. The new experimental function replaces S-boxes with simple χ functions.

Keywords: Hummingbird-2, Related-Key Cryptanalysis, Lightweight Cryptography, Authenticated Encryption, Hummingbird-2nu.

1 Introduction

Hummingbird-2 is a light-weight authenticated encryption primitive designed by a team led by Eric Smith of Revere Security and presented in RFIDSec '11 [8]. Hummingbird-2 has been proposed for standardization in RFID use within ISO [5].

Hummingbird-2 was created largely in response to an effective FSE '11 attack by Saarinen [14] against the original Hummingbird algorithm [6, 7, 10]. Saarinen's single-key attack broke the 256-bit Hummingbird-1 with 2^{64} effort.

Some independent analysis on Hummingbird-2 has been published. In [2] a "differential sequence attack" is described, but the total complexity of the attack is higher than exhaustive search and therefore it is "of theoretical interest only". The same is said of the side channel cube attack presented in [9]. An even more far-fetched attack is described in [20], requiring 2^{240} memory.

IACR ePrint [19] described an attack simultaneously using dozens of related keys. Unfortunately the attack, as described, had some errors and the authors subsequently withdrew the paper. However, some observations contained in it inspired our research that led to the discovery of high-probability correlated keys described in Section 2.1.

⋆ This research was sponsored by Revere Security (USA). Accepted to FSE 2013, March 11-13, 2013, Singapore. Preprint version 20130214093300.

The structure of this paper is as follows. In Section 2 we describe the relevant components of the Hummingbird-2 algorithm and make a number of observations about its various features. In Section 3 we describe an effective key-recovery attack that uses a single key relation. We discuss enabling factors of the attack in Section 3.7, followed by conclusions in Section 4.

Appendix A contains a full specification for a new variant which is resistant to these attacks and is based on novel χ functions (rather than traditional S-boxes).

2 Examining the Hummingbird-2 Algorithm

Hummingbird-2 is neither a block cipher nor a stream cipher in the traditional sense but combines some of the features of both. In this it resembles other integrated authenticated encryption proposals such as Helix [11] and Phelix [18].

The "Hummingbird structure" uses 16-bit data paths throughout as it was originally targeted towards low-end microcontrollers such as the TI MSP430 family. Data is always encrypted or decrypted in 16-bit increments. The cipher accepts a 64-bit initialization vector IV, a 128-bit secret key K, and maintains a 128-bit state in registers R. A method for deriving message authentication tags from the internal state is also given in the specification [8].

We use the following symbols and notation:

$x \oplus y$: Exclusive-or operation between x and y.
$x \boxplus y$: Modular addition $x + y \mod 2^{16}$.
$x \boxminus y$: Modular subtraction $x - y \mod 2^{16}$.
$x \lll n$: Left circular shift (rotation) of x by n bits.
$x \ggg n$: Right circular shift (rotation) of x by n bits.
S_i : A 4×4 - bit nonlinear substitution box, $i \in \{1, 2, 3, 4\}$.
IV_i : Word i of the 64-bit initialization vector, $i \in \{1, 2, 3, 4\}$.
K_i : Word i of the 128-bit secret key, $i \in \{1, 2, \cdots, 8\}$.
R_i^r : Word i of the 128-bit state at position r, $i \in \{1, 2, \cdots, 8\}$.
P^r, C^r : Plaintext and ciphertext words at position r.
$t_i^{(r)}$: Used to mark temporary, internal quantities.

In the following sections, we will describe the various algorithm components and present observations that will be used in the final overall attack. These cryptanalytic observations may also be useful in attacks of other types than the one described in this work. For a complete specification of Hummingbird-2, we refer the reader to [8].

2.1 WD16 (and High-Correlation Related Keys)

Hummingbird-2 draws almost all of its nonlinearity from the WD16 function. WD16 uses four keying words (total 64 bits) which define a permutation on a 16-bit input value. One may see WD16 as a 16-bit block cipher with a 64-bit key.

WD16 is a four-round substitution-permutation network. In each round, a 16-bit subkey is XORed to the state, four 4×4 - bit S-boxes are applied in parallel, followed by a linear mixing step. The structure is shown in Figure 1.

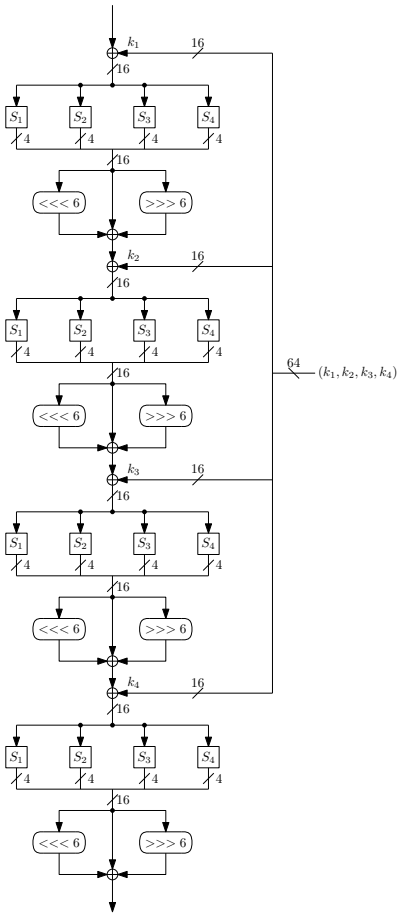

Fig. 1. The "WD16" mixing function is a 16-bit substitution-permutation network with four rounds and a 64-bit subkey (k_1, k_2, k_3, k_4). It is used in both initialization and encryption phases.

We use $S(x)$ to denote the parallel application of the 4-bit S-boxes S_1, S_2, S_3, S_4 on the 16-bit word x. The linear operation is $L(x) = x \oplus (x \lll 6) \oplus (x \ggg 6)$. If we shorten their compound operation to $LS(x) = L(S(x))$ then WD16 can be written as:

$$\mathrm{WD16}(x, k_1, k_2, k_3, k_4) = \mathrm{LS}(\mathrm{LS}(\mathrm{LS}(\mathrm{LS}(x \oplus k_1) \oplus k_2) \oplus k_3) \oplus k_4). \qquad (1)$$

We occasionally also use LS^{-1} and $\mathrm{WD16}^{-1}$ to denote the inverses of respective functions. We fist observe that the WD16 can produce closely correlated output with some distinct but related keys.

Observation 1 *Consider two 64-bit WD16 keys* (k_1, k_2, k_3, k_4) *and* (k'_1, k'_2, k'_3, k'_4) *that for some* $i \in \{1, 2, 3\}$ *are related by* $\delta = k_i \oplus k'_i$ *and* $\Delta = k_{i+1} \oplus k'_{i+1}$*, with the other two key words equivalent. There are such pairs that will yield equivalent WD16 encryption and decryption for approximately* $1/4$ *for input and output values.*

Table 1. All $4 \times 18 = 72$ high-probability related key word pairs where $\delta = k_i \oplus k'_i$ is canceled by $\Delta = k_{i+1} \oplus k'_{i+1}$ in the WD16 nonlinear function with probability 1/4.

$\delta \rightarrow \Delta$	$\delta \rightarrow \Delta$	$\delta \rightarrow \Delta$	$\delta \rightarrow \Delta$
0001 → 3B8E	0010 → 74D3	0100 → C30C	1000 → D374
0002 → 2A8A	0010 → DC71	0100 → C71D	2000 → 6198
0002 → 2ECB	0020 → 30C3	0200 → 4D37	2000 → E3B8
0003 → 0441	0020 → B8E3	0300 → 8208	3000 → 2088
0007 → 0441	0030 → CC30	0300 → 8E3B	3000 → B2EC
0007 → 3B8E	0040 → CC30	0400 → 4515	5000 → E3B8
0008 → 1545	0050 → 1041	0400 → 8619	6000 → 8220
0008 → 3FCF	0060 → DC71	0600 → 4926	7000 → 8220
0009 → 330C	0060 → FCF3	0700 → 0822	8000 → 9264
000A → 1104	0070 → 1041	0700 → 8E3B	8000 → C330
000A → 3FCF	0080 → 5451	0A00 → 8208	9000 → 5154
000B → 0882	00A0 → 4410	0B00 → 0411	B000 → 1044
000C → 0CC3	00B0 → FCF3	0B00 → 4926	B000 → B2EC
000C → 2208	00C0 → 6492	0C00 → 4104	C000 → 4110
000E → 0882	00D0 → 2082	0D00 → 4D37	E000 → 1044
000E → 2649	00D0 → B8E3	0E00 → 0411	E000 → F3FC
000F → 1DC7	00F0 → 4410	0E00 → CF3F	F000 → 4110
000F → 2649	00F0 → 5451	0F00 → 4104	F000 → 6198

In a differential attack we only want to have a single active S-box to maximize the probability. As with any 4×4 S-box, each one of S_1, S_2, S_3 and S_4 must have differentials that work for at least four of the 16 input values, leading to the given probability $1/4$.

Looking at Figure 1 we can see how after the $\delta = k_i \oplus k'_i$ difference is introduced at position i, it is then subjected to a S-box substitution and a a linear transformation before the $\Delta = k_{i+1} \oplus k'_{i+1}$ key difference cancels it out at $i + 1$ with the given probability 1/4.

Table 1 gives a list of all of such pairs that have the optimum probability of exactly 1/4. This table was created via an exhaustive search.

We give some examples of WD16 key pairs for which $WD16_A(x) = WD16_B(x)$ with probability 1/4:

```
A = 0001 0000 0000 0000    B = 0000 3B8E 0000 0000
A = FFFF FFFF F000 6198    B = FFFF FFFF 0000 0000
A = 1234 5000 6090 1234    B = 1234 A000 0108 1234
```

The last two examples use the F000 → 6198 relation which was (randomly) chosen for the main attack described in Section 3 of this paper. There is a wide spectrum of variations of a more general attack methodology that is represented by that specific case; picking some other relation leads to a different attack.

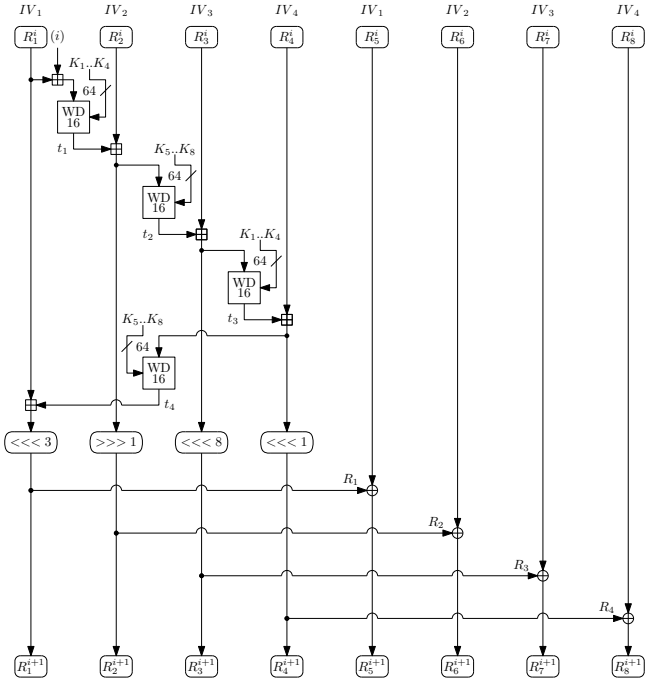

Fig. 2. Initialization round. There are four initialization rounds with a counter stepping through $i = 0, 1, 2, 3$.

2.2 Initialization and State Collisions

The initialization phase of Hummingbird-2 creates a 128-bit initial state from the 64-bit IV using the secret key and the WD16 function.

Initialization is a four-round process. Figure 2 shows a single initialization round. The state is first set as $R = IV \mid IV$. In each round, there are are four invocations of WD16 together with some $\bmod 2^{16}$ additive mixing, followed by cyclic rotations of the first four registers and linear exclusive-or "accumulation" mixing of the first four registers with the last four. The round counter $i = 0, 1, 2, 3$ is also used in the mix at the very beginning. The input keys to WD16 alter between the two halves of the master key (K_1, K_2, K_3, K_4) and (K_4, K_5, K_7, K_8).

Observation 2 *For each key K, there is a family of 432 related keys K' that yield the same state R after four initialization rounds with probability $P = 2^{-16}$ over all IV values.*

There are six possible positions i for $\delta = K_i \oplus K'_i$ and $\Delta = K_{i+1} \oplus K'_{i+1}$ that maximize the probability; $i \in \{1, 2, 3, 5, 6, 7\}$. Since there are two S-box activations in each round and four initialization rounds, the total probability of arriving at the same initial state for two such related keys is $(1/4)^{2\times 4} = 2^{-16}$. As there are 72 suitable (δ, Δ) pairs (see Table 1), for each 128-bit key K there are at least $6 \times 72 = 432$

related keys that will give the same initial state with the given 2^{-16} probability. This observation has been experimentally verified.

2.3 Encryption

Hummingbird-2 encrypts and decrypts data in 16-bit increments, as shown in Figure 3. The 128-bit state R^i and key K define a permutation from the plaintext word P^i to the ciphertext word C^i or vice versa. To encrypt plaintext word P^i into a ciphertext word C^i, the following steps are taken:

$$
\begin{aligned}
t_0^i &= P^i \boxplus R_1^i \\
t_1^i &= \mathrm{WD16}(t_0^i, K_1, K_2, K_3, K_4) \\
t_2^i &= \mathrm{WD16}(t_1^i \boxplus R_2^i, K_5 \oplus R_5^i, K_6 \oplus R_6^i, K_7 \oplus R_7^i, K_8 \oplus R_8^i) \\
t_3^i &= \mathrm{WD16}(t_2^i \boxplus R_3^i, K_1 \oplus R_5^i, K_2 \oplus R_6^i, K_3 \oplus R_7^i, K_4 \oplus R_8^i) \\
t_4^i &= \mathrm{WD16}(t_3^i \boxplus R_4^i, K_5, K_6, K_7, K_8) \\
C^i &= t_4^i \boxplus R_1^i.
\end{aligned}
$$

After each encrypted word is processed, the state is updated:

$$
\begin{aligned}
R_1^{i+1} &= R_1^i \boxplus t_3^i \\
R_2^{i+1} &= R_2^i \boxplus t_1^i \\
R_3^{i+1} &= R_3^i \boxplus t_2^i \\
R_4^{i+1} &= R_4^i \boxplus R_1^i \boxplus t_3^i \boxplus t_1^i \\
R_5^{i+1} &= R_5^i \oplus (R_1^i \boxplus t_3^i) \\
R_6^{i+1} &= R_6^i \oplus (R_2^i \boxplus t_1^i) \\
R_7^{i+1} &= R_7^i \oplus (R_3^i \boxplus t_2^i) \\
R_8^{i+1} &= R_8^i \oplus (R_4^i \boxplus R_1^i \boxplus t_3^i \boxplus t_1^i).
\end{aligned}
$$

For decryption, an inverse of WD16 function is required and the t quantities are computed in reverse order. The update function remains the same.

2.4 Related-Key Progression in Encryption

We see that there are four invocations of WD16 in each encryption operation and that key halves $K_1..K_4$ and $K_5..K_8$ are used twice each. In the middle two WD16 rounds the key is XORed with four of the higher "accumulator" state registers, but that has no effect on the differential. Since the differential is activated twice, there is a $(1/4)^2 = 1/16$ probability of matching ciphertexts.

Observation 3 *There is a 1/16 probability that for a matching state R the related keys K and K' (as defined in Section 2.1) will encrypt the same plaintext word to the equivalent ciphertext word.*

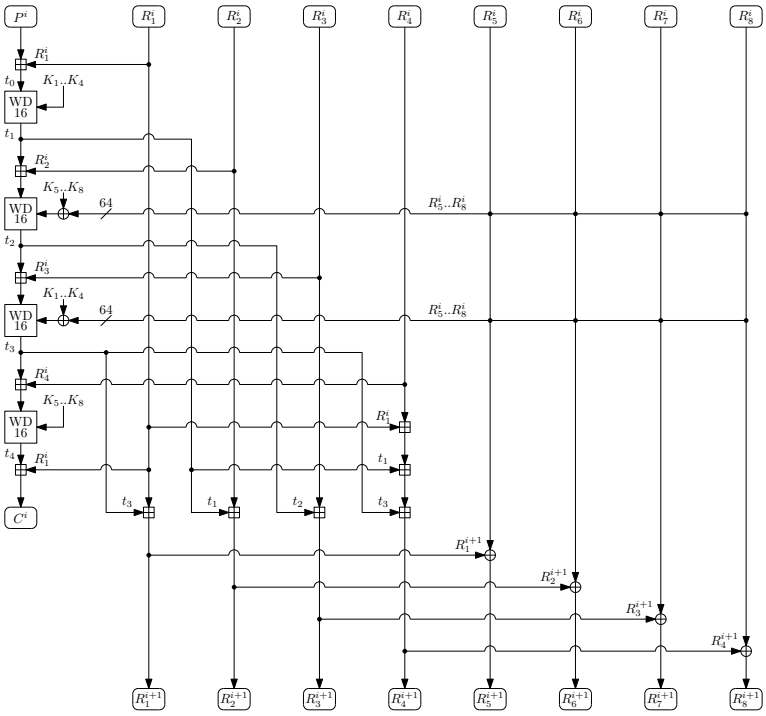

Fig. 3. Encryption of plaintext word P^i to ciphertext word C^i and update of state R. The "temporary" variables $t_0 \cdots t_4$ are used in the description of the attack.

Note that if the key difference is in $K_5..K_8$, there is a 1/4 probability of equivalent state update as the last WD16 invocation only affects ciphertext output, not the state. Conversely, if the key difference is in $K_1..K_4$, the state update will be equivalent in decryption with 1/4 probability. Furthermore, if the (δ, Δ) difference is in (K_1, K_2) as the first WD16 does not affect the state in decryption and at least 12 bits of the plaintext will be equivalent as there is only one active S-box.

3 Crafting an Attack

There are many ways that one can use the high-probability correlated keys in an attack. We will describe the one that we implemented, which uses only a single related key pair described in Section 3.1.

The attack proceeds in a number of distinct stages. We first find a suitable IV values for the attack (Section 3.2), and then proceed to solve various internal quantities (Sections 3.3 and 3.4) and finally parts of the secret key (Sections 3.5 and 3.6).

3.1 Attack Model

We assume that the attacker has access to two "black box" oracles whose keys are related by

$$K \oplus K' = (\texttt{F000 6198 0000 0000 0000 0000 0000 0000}). \tag{2}$$

The choice of this particular key relation is almost arbitrary in the set of admissible key dfferences. Many of the differentials in Table 1 could be used as well.

In our model the attacking algorithm may perform chosen-IV initializations and query encryptions and decryptions from the oracles. For an ideal cipher the most effective way to recover the secret key K (and K') would be to through brute force with expected complexity of 2^{128} trials. Therefore we will use the estimated time required for a single trial, consisting of initialization and encryption/decryption of a single word as the "unit complexity" $c = 2^0$.

We note that in a brute force attack eight words need to be encrypted in order to be reasonably sure that the correct key has been found, but with the probability 65535/65536 the incorrect ones can be rejected after encryption of a single word. Hence we use this as the unit complexity.

3.2 Finding a State Collision

The first stage of the attack is to find an IV value that produces a matching state R after the four-round initialization procedure for both K and K'. As indicated by Observation 2 in Section 2.2, one expects to find such a collision after searching through 2^{16} different IV values. Detection of a collision can be made by trial decryptions. If we decrypt a word x immediately after initialization, then there is a 1/4 probability that 12 bits of the corresponding plaintext words will match as discussed in Section 2.4. The overall complexity of this step is no more than 2^{20} to find an IV collision that holds with overwhelming probability.

Note that subsequent collisions may be found faster (for this K_1, K_2 relation) if we first search using words (IV_1, IV_2, IV_3) and for consecutive searches keep those words constant and loop through values of IV_4. The two initial round collisions are therefore guaranteed and consecutive collisions can be found with probability 2^{-12}.

Our attack requires only a single initialization state collision, henceforth denoted simply as IV.

3.3 Attacking R_1^i with Carry Bits

It is important to note that in HB2 encryption we can also have state and ciphertext word collisions when the plaintext words P (for K instance) and P' (for K' instance) are not equal.

The next stage involves the recovery of R_1^i. We can generate full codebooks $P^i \leftrightarrow C^i$ and $P'^i \leftrightarrow C'^i$ that depend on the IV and previous P^j, $j < i$ values with roughly 2^{17} effort if i is small. We fix $C^j = C'^j$ for $j < i$ and the states R^i do not diverge. Looking at Figures 1 and 3 we note the following.

Table 2. High nibbles of intermediate values $N = ((P^i \boxplus R_1^i) \oplus K_1)) >> 12$ and $N' = ((P'^i \boxplus R_1^i) \oplus K_1') >> 12$ in WD16 that will provide a collision. These are the pairs for which $S_1(N) \oplus S_1(N' \oplus \texttt{0xF}) = \texttt{0x6}$. Note that in the diagonal there are four entries as expected; if $N = N'$ there is a 1/4 probability of a collision.

$N \backslash N'$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	-	-	-	-	-	-	-	-	-	-	A	-	-	-	-	-
1	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	-	-	-	8	-	-	-	-	-	-	-
4	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-
5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F
6	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-	-	-	-	-	-	C	-	-	-
8	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-
9	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	7	-	-	-	-	-	-	-	-
B	-	-	-	-	-	-	6	-	-	-	-	-	-	-	-	-
C	-	-	-	-	-	-	-	-	-	-	B	-	-	-	-	-
D	-	-	-	-	-	-	-	-	-	-	-	-	D	-	-	-
E	-	-	-	-	-	-	-	-	-	-	-	-	-	E	-	-
F	-	-	-	-	-	-	-	-	-	9	-	-	-	-	-	-

Observation 4 *The first (δ, Δ) collision in the encryption operation works when*

$$S((P^i \boxplus R_1^i) \oplus K_1) \oplus S((P'^i \boxplus R_1^i) \oplus K_1') = L^{-1}(\Delta). \tag{3}$$

Here we use S to denote the four parallel S-box lookups and L^{-1} to denote the inverse of the shift/XOR linear step in WD16, as in Equation 1.

The δ and Δ values dictate which values the input differential $P^i \oplus P'^i$ can take. Since the input differential $\delta = K_1 \oplus K_1' = \texttt{F000}$ is in the high nibble, only the high nibbles $N = ((P^i \boxplus R_1^i) \oplus K_1)) >> 12$ and $N' = ((P'^i \boxplus R_1^i) \oplus K_1') >> 12$ really matter. We can tabulate successful pairs; see Table 2.

We see that Table 2 has only one entry per each horizontal and vertical line; N' can be given as a function of N and vice versa. If the N and N' entries are shifted by one position the collision at that point becomes impossible.

As we only want to have a single active S-box, may choose the high nibbles of P^i and P'^i arbitrarily, but we have to keep the low 12 bits the same.

Observation 5 *The probability of the carry shift depends solely on the value of plaintext low bits and the low bits of R_1^i. The shift will occur only when*

$$(P^i \wedge \texttt{0FFF}) + (R_1^i \wedge \texttt{0FFF}) \geq \texttt{1000}. \tag{4}$$

Since we have created a codebook of $P^i \leftrightarrow C^i$, we may effectively loop through the low 12 bits of $p = P^i \wedge \texttt{0FFF} = P'^i \wedge \texttt{0FFF}$ and until the carry-over "shift" occurs and the pattern changes from $p = \texttt{0000}$. This will give us the low bits of R_1^i. This

process isn't entirely foolproof as there are is a second collision that is required in the encryption process, but due to abundance of trials we may accurately pinpoint the p carry transition point with a good probability.

For each p value we may test $16 \times 16 = 256$ high nibble pairs for a matching ciphertext collision. Those collisions must occur at the points with an entry in Table 2. We may loop from low values of p towards higher values and see the lowest p value which starts to give different "grid". The algorithm we use is therefore essentially based on elimination of impossible combinations.

Note that the K_1 keying XOR in Equation 3 also affects this step and the actual shift that occurs. However, we have found that if we guess the highest bit of K_1 (and hence K_1' which has the inverse high bit), we can actually determine all 16 bits of R_1^i with high probability with roughly 2^{17} total complexity and one guessed bit.

3.4 Deriving Additional Quantities for an Attack

From Section 2.3 we see that R_1 is updated as $R_1^{i+1} = R_1^i \boxplus t_3^i$. If we have derived two consecutive R_1 values using the technique outlined in Section 3.3, we obtain the value of t_3 at round i:

$$t_3^i = R_1^{i+1} \boxminus R_1^i. \tag{5}$$

Furthermore, since $C^i = t_4^i \boxplus R_1^i$, we obtain

$$t_4^i = C^i \boxminus R_1^i. \tag{6}$$

This stage proceeds by attempting to create a sequence where $t_4^i = t_4^{i+1}$ holds with a high probability. To do this, for $i = 1, 2, 3 \cdots 2^7$ process each full 16-bit codebook as discussed in Section 3.3 and choose C_i to be the smallest value after R_1^i such that corresponding P_i and P_i' form a state collision.

For those pairs where $t_4^i = t_4^{i+1}$, the following relation holds since WD16 is a permutation and matching output words imply matching input words:

$$t_3^i \boxplus R_4^i = t_3^{i+1} \boxplus R_4^{i+1}. \tag{7}$$

We manipulate Equation 7 into $t_3^i = t_3^{i+1} \boxplus R_4^{i+1} \boxminus R_4^i$ and substitute that into the R_4 update function

$$R_4^{i+1} = R_4^i \boxplus R_1^i \boxplus t_3^i \boxplus t_1^i \tag{8}$$

to obtain

$$t_1^i = \boxminus R_1^i \boxminus t_3^{i+1}. \tag{9}$$

Since R_1^i and t_3^{i+1} are known quantities, as is $t_0^i = P^i \boxplus R_1^i$, we now can attack the first half of the keywords:

$$t_1^i = \mathrm{WD16}(t_0^i, K_1, K_2, K_3, K_4). \tag{10}$$

Note that due to the probabilistic nature of our R_1 derivation method, not all of these candidate pairs are valid. However, we have experimentally verified that in practice a sufficient number is valid and the key search algorithm (described in Section 3.5) is designed in a way that accounts for false pairs.

3.5 A Time-Memory Trade-off for $K_1 \cdots K_4$ Search

The information obtained in Sections 3.3 and 3.4 – especially Equation 10 – already allow the keyspace of Hummingbird-2 to be split in half and a 2^{64} attack can be mounted via exhaustive search. We will describe a simple time-memory tradeoff attack that allows further square root reduction for the first half of the key words.

In this step, we are given n values (x_i, y_i), $1 \leq i \leq n$, that satisfy

$$\mathrm{WD16}(x_i, K_1, K_2, K_3, K_4) = y_i \tag{11}$$

with a reasonable probability (see Equation 10).

We've experimentally discovered that if we perform the search for matching consecutive t_4 pairs discussed in Section 3.4 up to a limit of 2^7 plaintext / ciphertext words, we are typically left with $n = 2^4$ candidates. Out of these, about 2^3 will be "right pairs" that actually satisfy Equation 11 for the correct subkeys. This is a sufficient fraction for a time-memory trade-off technique.

To eliminate one of the keys, we pair the the values and investigate (x_i, y_i) and (x_j, y_j), $1 \leq i \leq j \leq n$. There are $n(n-1)/2$ pairs, quarter of which will be right pairs. This will help to cancel out K_3 in the computation.

Table Generation. For each i, j pair, we first construct a lookup table for subkey K_4. For each guessed $0 \leq K_4 < 2^{16}$ we compute the middle value h and build a table $T()$:

$$\begin{aligned} h &= \mathrm{LS}^{-1}(\mathrm{LS}^{-1}(y_i) \oplus K_4) \oplus \mathrm{LS}^{-1}(\mathrm{LS}^{-1}(y_j) \oplus K_4) \\ T(h) &= K_4. \end{aligned}$$

Here a candidate for K_4 can be obtained from the h value by building an appropriate data structure that takes care of collisions.

Key search. Approaching the WD16 from the other direction, we then loop through the 2^{32} values of K_1 and K_2 and look for a match in

$$h' = \mathrm{LS}(\mathrm{LS}(x_i \oplus K_1) \oplus K_2) \oplus \mathrm{LS}(\mathrm{LS}(x_j \oplus K_1) \oplus K_2) \tag{12}$$

Here $T(h')$ gives a candidate for K_4 with $O(1)$ effort. Then we check for *all* $1 \leq k \leq n$ pairs (x_k, y_k) how many of those yield the same K_3 value

$$K_3^? = \mathrm{LS}(\mathrm{LS}(x_k \oplus K_1) \oplus K_2) \oplus \mathrm{LS}^{-1}(\mathrm{LS}^{-1}(y_k) \oplus T(h')). \tag{13}$$

If five or six of those K_3 values agree, then there is a significant probability that we have found the correct 64-bit quartet (K_1, K_2, K_3, K_4) of the secret key words.

Complexity. Since about 2^4 lookup key searches of 2^{32} primitive operations (and a total of 2^{16} memory) is required, we estimate that the total complexity of this step is less than 2^{36} when adjusted to the scale of the complexity of brute force key search as discussed in the beginning of Section 3.

3.6 Finding the rest: $K_5 \cdots K_8$ Search

After the first half of the keying material has been discovered, it is a simple matter to brute force the rest. We have not found a time-memory tradeoff or other simple shortcut for the recovery of this part. Hence the total complexity is dominated by the second half, giving the total complexity of 2^{64} processing and about 2^{16} data.

It is quite easy to see that the last WD16 instance could be used to speed up key recovery if the difference between two keys would be at the right half of the key. However, in the beginning of Section 3 we chose a specific difference which lies at the first words. If we adopt the nonstandard setting of [19] where more than two "black boxes" with specific key relations can be accessed, then it the overall complexity of key recovery can be pushed down to the 2^{36} range. However, this attack model is rather unrealistic.

3.7 Discussion

Our attacks are specific to the Hummingbird structure as they do not purely follow any clear classical attack path such as linear or differential cryptanalysis. One may create a number of different attacks based on the same observations.

We developed the attack described in this paper while we were implementing it. One discovery led to the next. Our attack implementation used clear black box insulation and therefore we have a high degree of confidence that it works. We have tested it with various subsets of key space.

Design issues. The attacks are made possible by a combination of factors. Lessons were perhaps not fully learned from the attacks of [14] which exploited the simplistic key schedule and and algebraic properties of the the Hummingbird structure. However, a simple and fast key schedule is partly dictated by the timing constraints of the RFID environment and protocols for which Hummingbird was designed. It can also be argued that having 16-bit datapaths with additive mixing has certain advantages when a cipher is specifically to be used with a 16-bit embedded CPU, even though the particular structure of Hummingbird may not fully utilize the potential.

Fixing WD16: Hummingbird-2ν. The main enabler of the attacks is the WD16 function and the way it is keyed. Furthermore WD16 has a linear mixing stage $L(x)$ that has suboptimal diffusion and does not allow effective use of lookup tables to speed up decryption of data like the MDS [17] matrices of SHARK [13] and AES [12] do.

To mitigate both security and efficiency issues, we propose an alternative where $\mathrm{WD16}(x, k_1, k_2, k_3, k_4)$ has been replaced with "S-boxless" $\chi_\nu(x, k_1, k_2, k_3, k_4)$ to produce a variant called Hummingbird-2ν. Hummingbird-2ν is described in more detail in Appendix A. This variant is geared towards hardware implementation. We note that that the estimated implementation footprint for a 32-cycle version of HB2 is only 500 GE and an implementation that can perform both encryption and decryption is around 700 GE. More accurate implementation results will be reported separately.

4 Conclusions

We have discovered and demonstrated large related key classes which produce closely correlated output for any given input. The weak key classes penetrate both the initialization and actual ciphering stages of Hummingbird-2.

We have developed a full key recovery related-key attack algorithm which effectively halves the cipher's key size. This attack allows the secret key can to be recovered with only 2^{64} time and 2^{16} data in a two-key setting. The attack has been implemented and verified to work. Furthermore, the first half of the key can be recovered with only 2^{36} effort. Other types of attacks may be derived from the same observations.

Even though it may be tempting to derive multiple keys from a single one (e.g. one for each communication direction or medium), Hummingbird-2 should only be used with strictly random keys. This approach is taken in the ISO protocol proposal [5]. System designs where the secret keys of tags are related or shortened should be avoided. Key bits must never be used to denote access / product categories or other information.

References

1. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: The Keccak reference, version 3.0. NIST SHA3 Submission Document (January 2011)
2. Chai, Q., Gong, G.: A cryptanalysis of Hummingbird-2: The differential sequence analysis. IACR ePrint 2012/233, `http://eprint.iacr.org/2012/233` (April 2012)
3. Daemen, J.: Cipher and Hash Function Design Strategies based on linear and differential cryptanalysis. Ph.D. thesis (March 1995)
4. Ehrenfeucht, A., Mycielski, J.: A pseudorandom sequence - how random is it. Amer. Math. Monthly 99, 373–375 (1992)
5. Engels, D.: HB2-128 Crypto-Suite proposal. Tech. rep., Revere Security (December 2011), version 1.1
6. Engels, D., Fan, X., Gong, G., Hu, H., Smith, E.M.: Ultra-lightweight cryptography for low-cost RFID tags: Hummingbird algorithm and protocol. Tech. Rep. CACR-2009-29, University of Waterloo (2009), `http://www.cacr.math.uwaterloo.ca/techreports/2009/cacr2009-29.pdf`
7. Engels, D., Fan, X., Gong, G., Hu, H., Smith, E.M.: Hummingbird: Ultra-lightweight cryptography for resource-constrained devices. In: Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J.M., Sako, K., Sebé, F. (eds.) Financial Cryptography and Data Security, FC 2010 Workshops. LNCS, vol. 6054, pp. 3–18. Springer (2010)
8. Engels, D., Saarinen, M.J.O., Schweitzer, P., Smith, E.M.: The Hummingbird-2 lightweight authenticated encryption algorithm. In: Juels, A., Paar, C. (eds.) RFIDSec '11. LNCS, vol. 7055, pp. 19–31. Springer (2011)
9. Fan, X., Gong, G.: On the security of Hummingbird-2 against side channel cube attacks. In: Lucks, S., Armknecht, F. (eds.) WEWoRC 2011 – West European Workshop on Research in Cryptography. pp. 100–104 (2011)
10. Fan, X., Hu, H., Gong, G., Smith, E.M., Engels, D.: Lightweight implementation of Hummingbird cryptographic algorithm on 4-bit microcontroller. In: The 1st International Workshop on RFID Security and Cryptography (RISC'09). pp. 838–844 (2009)
11. Ferguson, N., Whiting, D., Schneier, B., Kelsey, J., Lucks, S., Kohno, T.: Helix: Fast encryption and authentication in a single cryptographic primitive. In: Johansson, T. (ed.) FSE 2003. LNCS, vol. 2887, pp. 330–346. Springer (2003)

12. NIST: Advanced Encryption standard (AES). Federal Information Processing Standards 197 (2001)
13. Rijmen, V., Daemen, J., Preneel, B., Bosselaers, A., Win, E.D.: The cipher SHARK. In: FSE '96. LNCS, vol. 1008, pp. 99–111. Springer (1996)
14. Saarinen, M.J.O.: Cryptanalysis of Hummingbird-1. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 328–341. Springer (2011)
15. Saarinen, M.J.O.: Cryptographic analysis of all 4 x 4 - bit s-boxes. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 118–133. Springer (2011)
16. Sutner, K.: The Ehrenfeucht-Mycielski sequence. In: Ibarra, O.H., Dang, Z. (eds.) Proc. Conf. Implementation and Application of Automata (CIAA 2003). LNCS, vol. 2759, pp. 73–82. Springer (2003)
17. Vaudenay, S.: On the need for multipermutations: Cryptanalysis of md4 and safer. In: FSE '94. LNCS, vol. 1008, pp. 286–297. Springer (1995)
18. Whiting, D., Schneier, B., Lucks, S., Muller, F.: Phelix – fast encryption and authentication in a single cryptographic primitive. ECRYPT Stream Cipher Project Report 2005/027 (2005), `http://www.schneier.com/paper-phelix.html`
19. Zhang, K., Ding, L., Guan, J.: Cryptanalysis of Hummingbird-2. IACR ePrint 2012/207, `http://eprint.iacr.org/2012/207` (April 2012)
20. Zhu, B., Gong, G.: Multidimensional meet-in-the-middle attack and its applications to GOST, KTANTAN and Hummingbird-2. IACR ePrint 2011/619, `http://eprint.iacr.org/2011/619` (November 2011)

A Hummingbird-2ν

The new experimental variant Hummingbird-2ν is the same as Hummingbird-2, expect that the WD16 substitution-permutation network has been replaced with a new function, $\chi_\nu(x, k_1, k_2, k_3, k_4)$. The new variant is geared towards hardware implementation and has a lower gate count than Hummingbird-2. Due to space constraints, we can only give a brief description of the new variant here and leave more detailed analysis for a separate report.

The new construction is based on χ functions, which are simple shift-invariant transformations that were first characterized by Daemen in [3]. The SHA3 algorithm Keccak uses a χ function as it's sole nonlinear component [1]. This selection was done in part to inspire research on functions of this type. The S-Boxes of the Hummingbird-2 WD16 design were selected based on extensive research [15].

We define two nonlinear functions f and g that operate on 16-bit words:

$$f(x) = \big((x \lll 2) \wedge \neg(x \lll 1) \wedge (x \ggg 1)\big) \oplus x$$
$$g(x) = \big(\neg x \wedge (x \lll 4) \wedge \neg(x \lll 12)\big) \oplus (x \lll 8)$$

The steps required to compute $y = \chi_\nu(x, k_1, k_2, k_3, k_4)$ are

$$t_1 = f(g(x \oplus k_1) \oplus \texttt{4D71}) \qquad t_2 = f(g(t_1 \oplus k_2) \oplus \texttt{0F65})$$
$$t_3 = f(g(t_2 \oplus k_3) \oplus \texttt{2746}) \qquad t_4 = f(g(t_3 \oplus k_4) \oplus \texttt{0B7C})$$
$$t_5 = f(g(t_4 \oplus k_1) \oplus \texttt{CFD5}) \qquad t_6 = f(g(t_5 \oplus k_3) \oplus \texttt{8E45})$$
$$t_7 = f(g(t_6 \oplus k_2) \oplus \texttt{40DA}) \qquad y = f(g(t_7 \oplus k_4) \oplus \texttt{62F0})$$

We acknowledge that one could use more of the keying material in each χ_ν function to make divide-and-conquer attacks more difficult. We decided not to change the overall structure outside the nonlinear component at all, however.

The "magic constants" `4D710F6527`.. are derived from the Ehrenfeucht-Mycielski sequence [4, 16]. The inverse function $x = \chi_\nu^{-1}(y, k_1, k_2, k_3, k_4)$ is easy to derive when we note that f and g are involutions: $f(f(x)) = x$ and $g(g(x)) = x$. The steps are simple performed in reverse order. In a hardware implementation the decryption circuitry closely matches the encryption circuit.

Here are some test vectors for χ_ν and a trace of execution for the last entry:

$$\chi_\nu(\texttt{0000}, \texttt{0000}, \texttt{0000}, \texttt{0000}, \texttt{0000}) = \texttt{FECB}$$
$$\chi_\nu(\texttt{1234}, \texttt{5555}, \texttt{5555}, \texttt{5555}, \texttt{5555}) = \texttt{18E6}$$
$$\chi_\nu(\texttt{0000}, \texttt{0123}, \texttt{4567}, \texttt{89AB}, \texttt{CDEF}) = \texttt{3286}$$

```
x=0000  t: 4C70 D80E 8857 2DB9 169D B89A 39B7  y=3286
```

Note that Hummingbird byte-word conversions are little-endian. Here's a test vector for Hummingbird-2ν encryption of 16 bytes and the resulting MAC:

```
KEY = 01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10
IV  = 12 34 56 78 9A BC DE F0

PT  = 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF
CT  = 63 66 F6 CB 60 0F A4 CE 52 78 D8 A8 5B 39 E2 B3

MAC = E8 50 64 50 68 CA 49 04 9C E8 6A 54 55 F0 00 F0
```