A Cryptanalysis of HummingBird-2: The Differential Sequence Analysis

Qi Chai and Guang Gong

Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, N2L 3G1, Canada
{q3chai, ggong}@uwaterloo.ca

Abstract. Hummingbird-2 is one recent design of lightweight block ciphers targeting constraint devices, which not only enables a compact hardware implementation and ultra-low power consumption but also meets the stringent response time as specified in ISO18000-6C.

In this paper, we present the first cryptanalytic result on the full version of this cipher using two pairs of related keys, i.e., four keys. We discover that the differential sequences for the last invocation of the round function can be computed by running the full cipher, due to which the search space for the key can be reduced. Base upon this observation, we propose a probabilistic attack encompassing two phases, preparation phase and key recovery phase. The preparation phase, requiring 2^{80} effort in time, aims to reach an internal state, with 0.5 success probability, that satisfies particular conditions. In the key recovery phase, by attacking the last invocation of the round function of the encryption (decryption resp.) using the proposed differential sequence analysis (DSA), we are able to recover 36 bits (another 44 bits resp.) of the 128-bit key. In addition, the remaining 48 bits of the key can be exhaustively searched and the overall time complexity of the key recovery phase is $2^{48.14}$.

Note that the proposed attack, though exhibiting an interesting tradeoff between the success probability and time complexity, is only of a theoretical interest at the moment and does not affect the security of the Hummingbird-2 in practice.

Keywords: lightweight cryptography, differential cryptanalysis, Hummingbird encryption

1 Introduction

Passive RFID tags and other constraint computing devices are usually characterized by extremely tight cost and power consumption requirements. The needs of cryptographic primitives on such devices have been increasing with the growing pervasiveness and mass deployment of these devices. To this end, considerable lightweight stream/block ciphers are proposed in recent years, targeting very small hardware footprint and reduced power consumption. Typical examples are listed in Table 1. Meanwhile, cryptanalysis of these lightweight primitives has received considerable attention due to a widely-accept concern – the pursue of efficiency at the cost of reducing the security margin or applying innovative but less well understood technologies lead lightweight candidates to be less durable relative to regular symmetric ciphers. This concern has been further confirmed by the successful cases of attacking KeeLoq [33], Crypto-I [32], Atmel Cipher [22, 9], PRESENT [13, 11], KTANTAN [7, 3], PRINTCipher [2, 29], reduced KLEIN [1], A2U2 [12] and so on.

Table 1. Recent Design/Implementation of Lightweight Ciphers (ordered by gate equivalent (GE))

	Key size[bits]	Block size[bits]	Area[GE]	Throughput[Kb/s]	Logic process[μm]
PRINTCipher-48 [26]	80	48	402	6.25	0.18
KTANTAN32 [14]	80	32	462	12.5	0.13
PRINTCipher-48 [26]	80	48	503	100	0.18
KTANTAN48 [14]	80	48	571	9.4	0.13
GOST [34]	256	64	651	24.24	0.18
Piccolo-80 [38]	80	64	683	14.8	0.13
KTANTAN64 [14]	80	64	684	8.4	0.13
LED-64 [21]	64	64	688	5.1	0.18
LED-128 [21]	128	64	700	3.4	0.18
PRINTCipher-96 [26]	160	96	726	3.13	0.18
Piccolo-128 [38]	128	64	758	12.1	0.13
KATAN32 [14]	80	32	802	12.5	0.13
KATAN48 [14]	80	48	916	9.4	0.13
PRINTCipher-96 [26]	160	96	967	100	0.18
KATAN64 [14]	80	64	1,027	8.4	0.13
PRESENT [35]	80	64	1,075	11.4	0.18
KLEIN-64 [20]	64	64	1,981	N/A	0.18
KLEIN-80 [20]	80	64	2,097	N/A	0.18
HummingBird-2 [17]	**128**	**16**	**2,159**	**N/A**	**0.13**
KLEIN-96 [20]	96	64	2,213	N/A	0.18
AES [19]	128	128	3,400	12.4	0.35

A Brief History of Hummingbird Cipher: Motivated by the design of the well-known Enigma machine, the first generation of Hummingbird (call it HB-1) was proposed by the engineers in Revere Security and was further analyzed and published in [16] as an ultra-lightweight cryptographic algorithm targeting low-cost RFID tags, smart cards, and wireless sensor nodes to meet the stringent response time and power consumption requirements. Although HB-1, with an innovative hybrid structure of block cipher and stream cipher, was designed to provide 256-bit security, Saarinen, in FSE'11, showed a chosen-IV and chosen-message attack [36] that can recover the full secret key with at most 2^{64} off-line computational effort under two related IVs. Recently, Revere Security published the second generation of Hummingbird (call it Hummingbird-2 or HB-2) in [17], which inherits the design philosophy from HB-1, e.g., it has a small block size of 16-bit to adapt the needs of encrypting short messages in RFID applications and it retains the hybrid structure as a security compensation for the small block size. High level differences between HB-1 and HB-2 are: (1) key size has been reduced to 128 bits to satisfy the actual need for constrained devices; (2) size of the internal state has been increased from 80 bits to 128 bits; (3) the nonlinear keyed transformation in HB-2 has four invocations of the S-boxes, compared to five in HB-1, to further increase the throughput.

In addition, it is claimed in the same paper that HB-2 can withstand differential, linear and algebraic attacks and the four 4-bit S-Boxes in HB-2 belong to the optimal classes as discussed in [31]. Its resistance to the side-channel cube attack is recently investigated in [18], where the author applied cube attack to recover 48 bits of the secret key providing the attacker could access the internal states of HB-2 during an early stage in the initialization. However, this attack is marginal since it only threats HB-2 before the finishing of its initialization.

Our Contribution: By refining/improving our preliminary results in [10], we present, in this paper, the first cryptanalytic result on the full version of this cipher using two pairs of related keys, i.e., four keys. Our attack makes use of the internal state of such a cipher and our philosophy is general: (1) the outputs of the encryption/decryption may leak information of the subkeys (under the differential cryptanalysis) as long as the internal states of the cipher satisfy particular conditions; (2) due to the birthday paradox, such a condition always happens with $1/2$ probability providing $2^{L/2}$ attempts are made, where L (in bit) is the size of the internal state. To be specific, we propose the following attack encompassing two phases, a probabilistic preparation phase and a key recovery phase.

- To realize the two particular conditions regarding the internal states, the preparation phase spends 2^{80} effort in time to achieve the success probability 0.5 (due to the birthday paradox). If succeeds, one could proceed to the key recovery phase.
- The key recovery phase is basically an instance of a novel cryptanalytic technique – we call it differential sequence analysis (DSA) – which can be seen as a hybrid of the conventional differential cryptanalysis and saturation attack. After exhibiting DSA's definitions and properties, we present its applications in the attacking scenarios, i.e.,
    - by using the encryption of HB-2, DSA recovers 36-bit (out of 128-bit) of the key, if condition (A) (regarding HB-2's secret key, input and internal state) holds.
    - by using the decryption of HB-2, DSA recovers another 44-bit of the key, if condition (B) (regarding HB-2's secret key, input and internal state) holds.
    - the rest 48-bit of the key can be exhaustively searched and the overall time complexity is $2^{48.14}$.
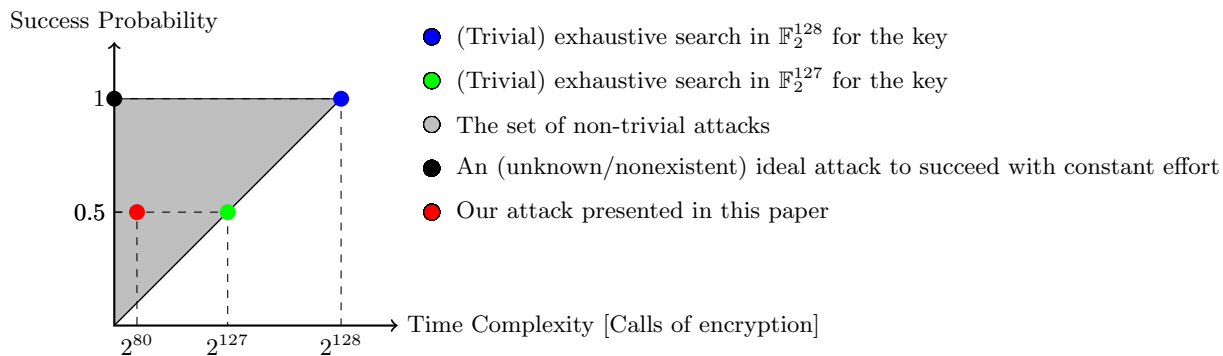
Fig. 1. Tradeoff between Success Probability and Time Complexity when attacking HB-2 (In fact, since one encryption only provides 16-bit entropy of the key, the exhaustive search needs a bit more than 2^{128} calls of the encryption function following the "key testing" procedure as described in [4], i.e., $2^{128}+2^{112}+2^{96}+2^{80}+2^{64}+2^{48}+2^{32}+2^{16} \approx 2^{128.000022}$ and 8 plaintext-ciphertext pairs to uniquely determine the key with probability 1.)

Note that our results in this paper exhibit an interesting tradeoff between the success probability and time complexity for HB-2, as shown in Fig. 1, which is analog to the collision attack in the hash function due to the birthday paradox. Stated in another way, to be successful with probability 0.5, our attack is faster than the exhaustive search (which is the best known) by a factor of 2^{48}. Unfortunately,

to succeed with probability 1, our preparation phase requiring more effort in time than the exhaustive search, which makes the proposed method only of theoretical interests at the moment, i.e., the attack presented in this paper does not affect the security of the Hummingbird-2 in practice.

Organization: In Section 2, the specification of HB-2 is presented. Section 3 describes the principle of our attack at a high level. In Section 4, we devise the DSA technique, discuss its properties and how to use it to attack parts of HB-2. In Section 5, we show how to achieve the desired conditions. We conclude the paper in Section 6.

Notations: Throughout the rest of this paper, we make use of the following notations.

- An hexadecimal number is indicated by a prefix "0x", e.g., 0x10 = 16.
- Unless otherwise stated, "+" denotes the addition in $\mathbb{F}_2$, which can also be vector-wise, e.g., $(a, b) + (c, d) = (a + c, b + d)$, where $a, b, c, d \in \mathbb{F}_2^m$.
- "$\boxplus$" or "$\boxminus$" operator denote addition or subtraction modulo 2^{16}.
- The high-bit XOR differential is defined as $H =$ 0x8000, a nice property of which is, given $x, x', y \in \mathbb{F}_2^{16}$ and $x + x' = H$, the following holds with probability 1,

$$(x \boxplus y) + (x' \boxplus y) = H, \quad (x \boxminus y) + (x' \boxminus y) = H, \quad (y \boxminus x) + (y \boxminus x') = H.$$

That is to say, as pointed out in [36], the differential H behaves the same under "+" and "$\boxplus/\boxminus$".

2 Specification of Hummingbird-2

Hummingbird-2 is a 16-bit block cipher with a 128-bit secret key $K = (K_1, ..., K_8) \in (\mathbb{F}_2^{16}, ..., \mathbb{F}_2^{16}) = \mathbb{F}_2^{128}$ and a 64-bit public initialization vector $IV = (IV_1, ..., IV_4) \in (\mathbb{F}_2^{16}, ..., \mathbb{F}_2^{16}) = \mathbb{F}_2^{64}$. As opposed to conventional block ciphers, it has an 128-bit internal state $R = (R_1, ..., R_8) \in (\mathbb{F}_2^{16}, ..., \mathbb{F}_2^{16}) = \mathbb{F}_2^{128}$, which participates in each encryption/decryption and is updated after that.

Building Block: $WD16 : \{0, 1\}^{16} \mapsto \{0, 1\}^{16}$ is the fundamental block or round function of HB-2 encryption, which is defined as

$$WD16(x, K_a, K_b, K_c, K_d) = f(f(f(f(x + K_a) + K_b) + K_c) + K_d),$$

where x is the varying input, e.g., plaintext, intermediate state, K_a, K_b, K_c, K_d are four 16-bit secret keys and the nonlinear function f is specified as

$$S(x) = S_1(x_1)||S_2(x_2)||S_3(x_3)||S_4(x_4), x = (x_1, x_2, x_3, x_4)$$
$$L(x) = x + (x <<< 6) + (x <<< 10)$$
$$f(x) = L(S(x)).$$

Note that the four S-boxes, i.e., $S_1(x_i)$ to $S_4(x_i)$, are given in Table 2.

Besides, the inverse of $WD16$ is employed in the decryption, which is defined as

$$WD16^{-1}(y, K_d, K_c, K_b, K_a) = f^{-1}(f^{-1}(f^{-1}(f^{-1}(y) + K_d) + K_c) + K_b) + K_a,$$

Table 2. S-boxes in HummingBird-2

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S_1(x)$	7	12	14	9	2	1	5	15	11	6	13	0	4	8	10	3	$S_1^{-1}(x)$	11	5	4	15	12	6	9	0	13	3	14	8	1	10	2	7
$S_2(x)$	4	10	1	6	8	15	7	12	3	0	14	13	5	9	11	2	$S_2^{-1}(x)$	9	2	15	8	0	12	3	6	4	13	1	14	7	11	10	5
$S_3(x)$	2	15	12	1	5	6	10	13	14	8	3	4	0	11	9	7	$S_3^{-1}(x)$	12	3	0	10	11	4	5	15	9	14	6	13	2	7	8	1
$S_4(x)$	15	4	5	8	9	7	2	1	10	3	0	14	6	12	13	1	$S_4^{-1}(x)$	10	7	6	9	1	2	12	5	3	4	8	15	13	14	11	0

where $y = WD16(x, K_a, K_b, K_c, K_d)$ and f^{-1} is the inverse of f. The four S-boxes used in f^{-1} are also listed in Table 2.

Initialization: Hummingbird-2 is initialized before use. Let $(R_1^{(r)}, ...R_8^{(r)}) \in \{0,1\}^{128}$ denote the internal state at the rth iteration in the initialization. The initialization can thus be formulated as, for $r = 0, 1, 2, 3$,

$$t_1 = WD16(R_1^{(r)} \boxplus <r>, K_1, K_2, K_3, K_4) \tag{1}$$

$$t_2 = WD16(R_2^{(r)} \boxplus t_1, K_5, K_6, K_7, K_8) \tag{2}$$

$$t_3 = WD16(R_3^{(r)} \boxplus t_2, K_1, K_2, K_3, K_4) \tag{3}$$

$$t_4 = WD16(R_4^{(r)} \boxplus t_3, K_5, K_6, K_7, K_8) \tag{4}$$

$$R_1^{(r+1)} = (R_1^{(r)} \boxplus t_4) \lll 3 \tag{5}$$

$$R_2^{(r+1)} = (R_2^{(r)} \boxplus t_1) \lll 1 \tag{6}$$

$$R_3^{(r+1)} = (R_3^{(r)} \boxplus t_2) \lll 8 \tag{7}$$

$$R_4^{(r+1)} = (R_4^{(r)} \boxplus t_3) \lll 1 \tag{8}$$

$$R_5^{(r+1)} = R_5^{(r)} + R_1^{(r+1)} \tag{9}$$

$$R_6^{(r+1)} = R_6^{(r)} + R_2^{(r+1)} \tag{10}$$

$$R_7^{(r+1)} = R_7^{(r)} + R_3^{(r+1)} \tag{11}$$

$$R_8^{(r+1)} = R_8^{(r)} + R_4^{(r+1)}, \tag{12}$$

where $<r>$ represents a counter and $(R_1^{(0)}, ..., R_8^{(0)}) = (IV_1, IV_2, IV_3, IV_4, IV_1, IV_2, IV_3, IV_4)$.

Note that R_5, R_6, R_7, R_8 do not participate in the randomization, i.e., Eq. (6)-(9), but simply XOR the historical statuses of R_1, R_2, R_3, R_4 respectively (behaving like XOR-MAC). This fact may nullify their contribution to the overall cryptanalytic strength of HB-2 under a side-channel injection attack – 64 injections and 64 invocations of HB-2 encryption are needed to recover (R_5, R_6, R_7, R_8). Details are provided in Appendix A.

Encryption: After the initialization, each encryption, by invoking the round function for four times, transforms a single plaintext word $P_i \in \mathbb{F}_2^{16}, i = 1, 2, ...,$ to a corresponding ciphertext word C_i, i.e.,

$$t_1 = WD16(R_1^{(i)} \boxplus P_i, K_1, K_2, K_3, K_4) \tag{13}$$

$$t_2 = WD16(R_2^{(i)} \boxplus t_1, K_5 + R_5^{(i)}, K_6 + R_6^{(i)}, K_7 + R_7^{(i)}, K_8 + R_8^{(i)}) \tag{14}$$

$$t_3 = WD16(R_3^{(i)} \boxplus t_2, K_1 + R_5^{(i)}, K_2 + R_6^{(i)}, K_3 + R_7^{(i)}, K_4 + R_8^{(i)}) \tag{15}$$

$$C_i = WD16(R_4^{(i)} \boxplus t_3, K_5, K_6, K_7, K_8) \boxplus R_1^{(i)}, \tag{16}$$

where $(R_1^{(i)}, ..., R_8^{(i)}) \in \mathbb{F}_2^{128}$ is the internal state during the ith encryption and it is updated, at the end of the encryption, as follows:

$$R_1^{(i+1)} = R_1^{(i)} \boxplus t_3 \tag{17}$$

$$R_2^{(i+1)} = R_2^{(i)} \boxplus t_1 \tag{18}$$

$$R_3^{(i+1)} = R_3^{(i)} \boxplus t_2 \tag{19}$$

$$R_4^{(i+1)} = R_4^{(i)} \boxplus t_1 \boxplus R_1^{(i+1)} \tag{20}$$

$$R_5^{(i+1)} = R_5^{(i)} + R_1^{(i+1)} \tag{21}$$

$$R_6^{(i+1)} = R_6^{(i)} + R_2^{(i+1)} \tag{22}$$

$$R_7^{(i+1)} = R_7^{(i)} + R_3^{(i+1)} \tag{23}$$

$$R_8^{(i+1)} = R_8^{(i)} + R_4^{(i+1)} \tag{24}$$

A shorthand of Eq. (13)-(24) is $C_i = E(P_i, K) = E(P_i, (K_1, ..., K_8))$.

Decryption: Decryption of a single word $C_i \in \mathbb{F}_2^{16}, i = 1, 2, ...,$ followed by the same initialization, is

$$u_3 = WD16^{-1}(C_i \boxminus R_1^{(i)}, K_8, K_7, K_6, K_5) \tag{25}$$

$$u_2 = WD16^{-1}(u_3 \boxminus R_4^{(i)}, K_4 + R_8^{(i)}, K_3 + R_7^{(i)}, K_2 + R_6^{(i)}, K_1 + R_5^{(i)}) \tag{26}$$

$$u_1 = WD16^{-1}(u_2 \boxminus R_3^{(i)}, K_8 + R_8^{(i)}, K_7 + R_7^{(i)}, K_6 + R_6^{(i)}, K_5 + R_5^{(i)}) \tag{27}$$

$$P_i = WD16^{-1}(u_1 \boxminus R_2^{(i)}, K_4, K_3, K_2, K_1) \boxminus R_1^{(i)}. \tag{28}$$

After this, the internal states are updated as in the encryption, i.e., using Eq. (17)-(24), where $t_3 = u_3 \boxminus R_4^{(i)}$, $t_2 = u_2 \boxminus R_3^{(i)}$ and $t_1 = u_1 \boxminus R_2^{(i)}$.

3 Overview of Our Cryptanalytic Method on the Full **HB-2**

Adversary Model: We consider a scenario that two paralleled executions of encryptions are $C_i = E(P_i, K)$ and $C_{i'}' = E(P_{i'}', K')$, where the internal states are $(R_1^{(i)}, ..., R_8^{(i)})$ and $(R'_1^{(i')}, ..., R'_8^{(i')})$ respectively, the intermediate values are (t_1, t_2, t_3) and (t_1', t_2', t_3') respectively, and K and K' are related. (Similar for the decryption). The attacker follows the chosen plaintext/ciphertext model such that the

attacker is free to choose plaintext $P_i \in \mathbb{F}_2^{16}$ and $P'_{i'} \in \mathbb{F}_2^{16}$, launch encryption without knowing the related keys, and observe the corresponding $C_i \in \mathbb{F}_2^{16}$ and $C'_{i'} \in \mathbb{F}_2^{16}$; or chooses $C_i \in \mathbb{F}_2^{16}$ and $C'_{i'} \in \mathbb{F}_2^{16}$, launches decryption without knowing the related keys, and observes the corresponding $P_i \in \mathbb{F}_2^{16}$ and $P'_{i'} \in F_2^{16}$.

Attack In A Nutshell: Block ciphers are usually based on iterating a cryptographically weak function sufficient number of times without disturbing, e.g., modifying, the outputs of intermediate rounds except whitening them with round-keys. Our attack on the full HB-2 exploits the fact that the internal states, instead of enhancing the overall cryptanalytic strength, give the attacker an opportunity to create an input differential for the last invocation of $WD16$ ($WD16^{-1}$ resp.) in the encryption (decryption resp.) and to retrieve the corresponding distribution of the output differences (call the collection of them a *differential sequence*), which is information-rich in a subset of key. Henceforth, after obtaining such a *template sequence*, the attacker, in an off-line environment, could search for the key bits associated, which usually constitute a subset of entire key bits. In all, our full attack can be divided into two phases: **preparation phase** as described in Section 5 and **key recovery phase** as described in Section 4.

Key Recovery Phase: In the key recovery phase, to remove the undesired interference introduced by the varying internal states when consecutive words are encrypted/decrypted, our attack here targets a specific encryption/decryption after the *preparation*, i.e., ith encryption/decryption for one HB-2 instance and i'th encryption/decryption for the other one. This is because given the key, IV, and the plaintext chain fed are fixed, the ith internal state and the i'th internal state are fixed as well. Henceforth, we omit the superscript/subscript i and i' of HB-2 variables for convenience when describing operations in the key recovery phase.

Providing the preparation phase succeeds, the attacker accomplishes the following utilizing the properties of the differential sequence analysis:

- Step 1. 36 bits of $(K_5, ..., K_8) \in \mathbb{F}_2^{64}$ are recovered using the differential sequence obtained from the last invocation of $WD16$ in the encryption if a particular condition meets, as shown in Fig. 2.
- Step 2. 28 bits of $(K_4, ..., K_1) \in \mathbb{F}_2^{64}$ are recovered using the differential sequence obtained from the last invocation of $WD16$ in the decryption if another particular condition meets.
- Step 3. the rest 64-bit key are exhaustively searched using either encryption or decryption.

To be specific, the condition needed to launch Step 1 in *key recovery phase* is:

$$\text{Condition (A)}: \quad \Delta K = (K_1, ..., K_8) + (K'_1, ..., K'_8) = (H, 0, 0, 0, H, 0, 0, 0)$$
$$\Delta P = P + P' = H$$
$$\Delta R = (R_1, ..., R_8) + (R'_1, ..., R'_8) = (0, 0, 0, 0, H, 0, 0, 0).$$

The condition needed to launch Step 2 in *key recovery phase* is:

$$\text{Condition (B)}: \quad \Delta K = (K_1, ..., K_8) + (K'_1, ..., K'_8) = (0, 0, 0, H, 0, 0, 0, H)$$
$$\Delta C = C + C' = H$$
$$\Delta R = (R_1, ..., R_8) + (R'_1, ..., R'_8) = (0, 0, 0, 0, 0, 0, 0, H).$$
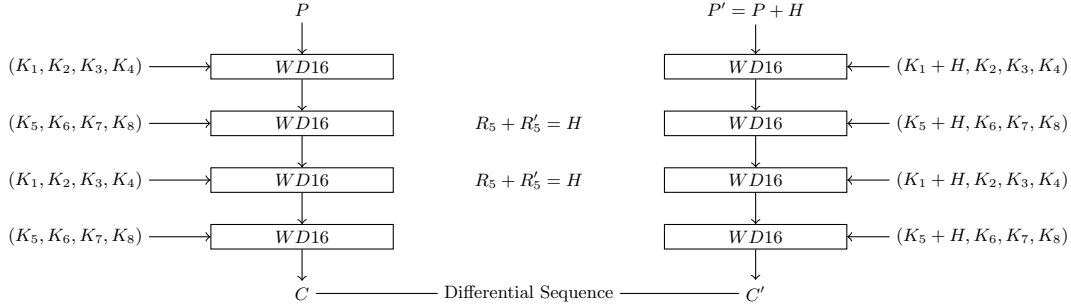
Fig. 2. Constructing Differential Sequence from Encryption with Condition (A)

To reach ΔP or ΔC in the two conditions above, the adversary model already allows the plaintext/ciphertext to be freely chosen; to reach ΔK, two pair of related-keys have to be used in our attack; and to reach ΔR, an extra phase, called preparation phase, has to be introduced.

Preparation Phase: As one may expect, preparation phase of our attack copes with the realization of ΔRs one at a time. To this end, one obvious way is to mount side-channel injection attack as shown in Appendix D, which gives the attacker no time/memory penalty, i.e., the overall time/memory complexity of the attack is dominated by that of the key recovery phase.

However, side-channel injection attack is not considered much in this work. Instead, we realize both conditions in a probabilistic manner, i.e.,

- $(R_1^{(i)}, ..., R_8^{(i)})$ and $(R_1'^{(i')}, ..., R_8'^{(i')})$ can be "randomized" by feeding both HB-2 instances with either different IVs and/or chains of random plaintext words. According to the birthday paradox, there is at least 0.5 chance that the randomized $(R_1^{(i)}, ..., R_8^{(i)}) \in \mathbb{F}_2^{128}$ and the randomized $(R_1'^{(i')}, ..., R_8'^{(i')}) \in \mathbb{F}_2^{128}$ satisfies ΔR in condition (A) (condition (B) resp.) providing 2^{64} attempts are made.
- Note that, in the previous step, even if ΔR happens, the attacker is usually unaware. To determine, we improve the mechanism above in light of another characteristic of HB-2, i.e., if condition (A) (condition (B) resp.) holds at the current round, it also holds for the next round. Hence, the differential sequences produced at the current round by $((R_1^{(i)}, ..., R_8^{(i)}), (R_1'^{(i')}, ..., R_8'^{(i')}))$ is exactly the same as that produced at the next round by $((R_1^{(i+1)}, ..., R_8^{(i+1)}), (R_1'^{(i'+1)}, ..., R_8'^{(i'+1)}))$.
- If the above step succeeds, the attacker proceeds to the key recovery phase to attack.

In what follows, we detail each of the above phases and steps.

4 Differentials Sequence Analysis (DSA)

In this section, we present a novel technique called differential sequence analysis (DSA) rooted in the differential cryptanalysis and the saturation attack. To be specific, we exhibiting its definitions, properties, and applications to attack one round of the HB-2, that constitutes the *key recovery phase* in our whole attack.

4.1 Differential Cryptanalysis and Saturation Attack

Differential cryptanalysis is a method analyzing the effect of particular differences in plaintext pairs on the differences of the resultant ciphertext pairs, which is based on a crucial observation that for any particular input differential, not all the output differential are possible, and the possible ones may not appear uniformly. In the original version of differential cryptanalysis [37], a unique differential is exploited to recover the subkey used in the last round of a block cipher. This idea has been extended in several ways: Biham and Shamir themselves further considered in [37] to use a trail of differentials to attack; Lai in [27] connected differential cryptanalysis with derivative of polynomials and presented a rigorous definition of higher order differentials; Knudsen [25] considered to use part of the input and output that have differential characteristics for the analysis; Biham, Biryukov and Shamir proposed in [5] to use differentials that happen with probability 0 as distinguishers; and recently, Blondeau and Gérard demonstrated the multiple differential cryptanalysis in [6], where a set of input/output differentials are considered together.

Saturation attack [28, 23, 8] exploits the fact that the output set is saturated, i.e., the outputs form the whole space of $\mathbb{F}_2^m$, if the input set for the m-bit core injective function is saturated. Since the saturation of the outputs is observable, this technique usually serves as a distinguisher for the attacker.

At a high level, our differential sequence analysis in this paper can be understood as a hybrid of the conventional differential cryptanalysis and saturation attack, i.e., the set of the output differentials (instead of the outputs themselves) with respect to a particular/fixed input differential and a saturated set of inputs is considered. From another angle, due to the use of output differentials caused by a saturated set of inputs, our attack is also a special case of multiple differential cryptanalysis [6].

4.2 (First-order) Differential Sequence

Assume we have a keyed permutation $h(w, K)$ mapping $w \in \mathbb{F}_2^m$ to $h(w, K) \in \mathbb{F}_2^m$ with respect to the secret key K, where m is a positive integer. Given a fixed $\theta \in \mathbb{F}_2^m$, the first-order differential is known as

$$\Delta_{\theta,K}(w) = h(w, K) + h(w + \theta, K).$$

The *(first-order) differentials sequence* of h at θ is basically one row in the differential distribution table of h with respect to the input differential θ. To discuss its properties, we define it in a more formal way.

Definition 1. *The first-order differential sequence (DS) of h at θ is a non-binary sequence of 2^m entries, i.e.,*

$$\Delta_{\theta,K} = (z_0, z_1, ..., z_{2^m-1}),$$

where z_i denotes the multiplicity (that is, number of occurrences) of i in the set $\{w \in \mathbb{F}_2^m | \Delta_{\theta,K}(w)\}$, i.e.,

$$z_i = |\{w \in \mathbb{F}_2^m | \ \Delta_{\theta,K}(w) = i\}|.$$

Note that this definition can be extended to higher orders. In this paper, we restrict ourselves to the first-order case.

For example, the differential sequence is $\{0, 0, 0, 2, 0, 0, 2, 0, 0, 4, 2, 0, 4, 0, 0, 2\}$ providing $\{w = \mathbb{F}_2^4 | \Delta_{\theta,K}(w)\} = \{12, 10, 3, 9, 6, 9, 15, 12, 12, 10, 3, 9, 6, 9, 15, 12\}$ and $\theta = 0x08$. The length of the differential sequence is the sum of all its multiplicities (16 in this example).

4.3 Properties of the Differential Sequence

The saturated set of inputs brings quite a lot of interesting properties to the conventional differential cryptanalysis. We list the core properties to attack HB-2 here.

Property 1. For a fixed $\theta \in \mathbb{F}_2^m$, $\Delta_{\theta,K}$ is constructed by evaluating and counting $(h(w,K)+h(w+\theta,K))$ for every w in $\mathbb{F}_2^m$ regardless of the order of $w \in \mathbb{F}_2^m$ been accessed.

This property follows immediately from Definition 1 and is useful in the sense that even though $h(w,K)$ is an intermediate round in a cipher (thus, w is an intermediate value), we are able to capture $\Delta_{\theta,K}$ given that θ can be fixed in a particular way and w traverses the whole space of $\mathbb{F}_2^m$. Stated in another way, we have the property below.

Property 2. Let $perm(w)$ be a permutation of w in $\mathbb{F}_2^m$, i.e., $perm(w) : \mathbb{F}_2^m \mapsto \mathbb{F}_2^m$. For a fixed $\theta \in \mathbb{F}_2^m$ and every $w \in \mathbb{F}_2^m$, $\Delta_{\theta,K}$ can be obtained either by evaluating and counting $(h(w,K) + h(w + \theta, K))$, or by evaluating and counting $(h(perm(w),K) + h(perm(w) + \theta, K))$.

In what follows, we use

$$[h(w,K) + h(w+\theta,K)|w \in \mathbb{F}_2^m] = [h(perm(w),K) + h(perm(w)+\theta,K)|w \in \mathbb{F}_2^m]$$

as a symbolic expression for Property 2, where $[...]$ actually defines a multiset and θ is always a fixed value in $\mathbb{F}_2^m$ for the rest of the paper. Henceforth, a straightforward extension of Property 2 can be derived below.

Property 3. Let $perm_i$, $i = 1, ..., n$, be permutations in $\mathbb{F}_2^m$. We have

$$\begin{aligned}
&[h(w,K) + h(w+\theta,K)|w \in \mathbb{F}_2^m] \\
&= [h(perm_n(...(perm_1(w))),K) + h(perm_n(...(perm_1(w))) + \theta, K)|w \in \mathbb{F}_2^m].
\end{aligned}$$

Proof. $perm_n(...(perm_1(w)))$ can be written as $perm(w)$ in $\mathbb{F}_2^m$. $\qquad\square$

As aforementioned, the obtained differential sequence is primarily used to search for the key bits associated. Henceforth, we are especially interested in the correspondences between the differential sequence and the K in the underlying function $h(w,K)$, e.g., is the mapping from K to the differential sequence injective or not? To this end, we start with a special case of Property 2.

Property 4. Providing $K = K_a \bigcup K_b$, $K_a \bigcap K_b = \emptyset$ and $h(w,K) = h(w + K_a, K_b)$, we have

$$[h(w,K) + h(w+\theta,K)|w \in \mathbb{F}_2^m] = [h((w+K_a),K_b) + h((w+K_a)+\theta,K_b)|w \in \mathbb{F}_2^m].$$

Proof. By applying Property 2 and set $perm(w) = w + K_a$, this property follows immediately. $\qquad\square$

From the property above, it is clear that all $K_a \in \mathbb{F}_2^{|K_a|}$ produces the same sequence while different K_bs may produce different sequences. Therefore, this property in fact implies that the obtained differential sequence of h at θ can be used to search for (a subset of) the key nonlinearly associated. Besides, there exists a more complicated correspondence between the key and the differential sequence. To discuss, we need to investigate the properties of sub-differential sequences.

Property 5. Let Γ be a subset of $\mathbb{F}_2^m$ and *perm* is a permutation in Γ, i.e., $perm(w) : \Gamma \mapsto \Gamma$, we have

$$[h(w, K) + h(w + \theta, K)|w \in \Gamma] = [h(perm(w), K) + h(perm(w) + \theta, K)|w \in \Gamma].$$

Proof. This property follows from Definition 1, if and only if *perm* is a permutation in Γ. □

Furthermore, we call $[h(w, K) + h(w+\theta, K)|w \in \Gamma]$ or $[h(perm(w), K) + h(perm(w) + \theta, K)|w \in \Gamma]$ a *sub-differential sequence* of $\Delta_{\theta, K}$. Therefore, we can actually view a differential sequence obtained in $\mathbb{F}_2^m$ as a summation of several sub-differential sequences obtained in the disjoint subspaces of $\mathbb{F}_2^m$. This intuition can be written as below.

Property 6. Let Γ_i, $i = 1, ..., q$, be q disjoint partitions of $\mathbb{F}_2^m$, i.e.,

$$\Gamma_i \cap \Gamma_j = \emptyset, \quad 1 \le i \ne j \le q \tag{29}$$

$$\cup_{i=1}^q \Gamma_i = \mathbb{F}_2^m \tag{30}$$

and let the differential sequence obtained by $[h(w, K) + h(w + \theta, K)|w \in \Gamma_i]$ be $\Delta_{\theta, K}^{\{\Gamma_i\}}$, we thus have,

$$\Delta_{\theta, K} = \sum_{i=1}^q \Delta_{\theta, K}^{\{\Gamma_i\}}.$$

Following this reasoning, Property 4 can also be extended as below, which tells us that a differential sequence in Γ only corresponds to the key nonlinearly (in Γ) associated.

Property 7. Providing $K = K_a \bigcup K_b$, $K_a \bigcap K_b = \emptyset$ and $h(w, K) = h(w + K_a, K_b)$, we have, if Γ is a subset of $\mathbb{F}_2^m$ and $(w + K_a)$ is a permutation in Γ with respect to K_a,

$$[h(w, K) + h(w + \theta, K)|w \in \Gamma] = [h((w + K_a), K_b) + h((w + K_a) + \theta, K_b)|w \in \Gamma].$$

Therefore, if each of the sub-differential sequence stays the same with respect to the keys belonging to a particular set, denoted as Φ_0, the overall differential sequence remain the same under Φ_0. We formalize this correspondence as below.

Property 8. Let $\Phi_0 = \cap_{i=1}^q \{k|w + k : \Gamma_i \mapsto \Gamma_i, k, w \in \mathbb{F}_2^m\}$, $K = K_a \bigcup K_b$, $K_a \bigcap K_b = \emptyset$ and $h(w, K) = h(w + K_a, K_b)$, we have

$$\Delta_{\theta, K} = \Delta_{\theta, \kappa}, \tag{31}$$

where $\kappa = \kappa_a \bigcup \kappa_b$, $\kappa_a \bigcap \kappa_b = \emptyset$, $K_a, \kappa_a \in \Phi_0$ and $\kappa_b = K_b$.

Proof. Let $\Delta_{\theta, K}^{\{\Gamma_i\}}$ be the sub-differential sequence obtained by Property 7. Thanks to Property 6, we have $\Delta_{\theta, K} = \sum_{i=1}^q \Delta_{\theta, K}^{\{\Gamma_i\}}$ and $\Delta_{\theta, \kappa} = \sum_{i=1}^q \Delta_{\theta, \kappa}^{\{\Gamma_i\}}$. Thanks to Property 7, for each i , $\Delta_{\theta, K}^{\{\Gamma_i\}} = \Delta_{\theta, \kappa}^{\{\Gamma_i\}}$ providing $K_a, \kappa_a \in \Phi_0$ and $\kappa_b = K_b$.

As opposed, providing $\kappa_b \ne K_b$ while $K_a, \kappa_a \in \Phi_0$, it is quite likely that $\Delta_{\theta, K} \ne \Delta_{\theta, \kappa}$ since $\Delta_{\theta, K}^{\{\Gamma_i\}} \ne \Delta_{\theta, \kappa}^{\{\Gamma_i\}}$ for each i.

□

4.4 Differential Sequence Analysis against HB-2

In this subsection, we attack the last invocation of $WD16$ ($WD16^{-1}$ resp.) in the encryption (decryption resp.) of HB-2 by exploiting the DSA as presented. To be specific, our Theorem 1 and Theorem 3 give answers to the question "how to obtain the differential sequences" while our Theorem 2 and Theorem 4 exhibit "how to use the differentials sequences". Since the HB-2 has a 16-bit block size, we have $m = 16$ for the rest.

Attacking $WD16$ in Encryption: To show our idea in a concise way, we assume that R_1 and R'_1 are known (in fact, they are identified by our algorithms in the preparation phase). In addition, let h in Definition 1 be the last invocation of $WD16$, i.e., Eq. (16), in the encryption. We thus have the following theorems.

Theorem 1. *When condition (A) meets, the differential sequence of the last $WD16$ in the encryption at $\theta = H$ can be extracted from executing the entire encryption.*

Proof: First of all, when condition (A) holds, we have,

$$
\begin{aligned}
t'_1 &= WD16(R'_1 \boxplus P', K'_1, K'_2, K'_3, K'_4) \\
&= WD16(R_1 \boxplus (P + H), (K_1 + H), K_2, K_3, K_4) = t_1 \\
t'_2 &= WD16(R'_2 \boxplus t'_1, K'_5 + R'_5, K'_6 + R'_6, K'_7 + R'_7, K'_8 + R'_8) \\
&= WD16(R_2 \boxplus t_1, (K_5 + H) + (R_5 + H), K_6 + R_6, K_7 + R_7, \\
&\quad K_8 + R_8) = t_2 \\
t'_3 &= WD16(R'_3 \boxplus t'_2, K'_1 + R'_5, K'_2 + R'_6, K'_3 + R'_7, K'_4 + R'_8) \\
&= WD16(R_3 \boxplus t_2, (K_1 + H) + (R_5 + H), K_2 + R_6, K_3 + R_7, \\
&\quad K_4 + R_8) = t_3
\end{aligned}
$$

Next, $\Delta_{H,(K_5,K_6,K_7,K_8)} = [z_0, z_1, ..., z_{2^{16}-1}]$ can be extracted, where

$$
\begin{aligned}
z_i &= |\{t_3 \in \mathbb{F}_2^{16}| \; (WD16(R_4 \boxplus t_3, K_5, K_6, K_7, K_8) + WD16(R'_4 \boxplus t'_3, K'_5, K'_6, K'_7, K'_8)) = i\}| \\
&= |\{t_3 \in \mathbb{F}_2^{16}| \; (WD16(R_4 \boxplus t_3, K_5, K_6, K_7, K_8) + WD16(R_4 \boxplus t_3, (K_5 + H), K_6, K_7, K_8)) = i\}| \\
&= |\{t_3 \in \mathbb{F}_2^{16}| \; (WD16(R_4 \boxplus t_3, K_5, K_6, K_7, K_8) + WD16((R_4 + H) \boxplus t_3, K_5, K_6, K_7, K_8)) = i\}| \\
&= |\{P \in \mathbb{F}_2^{16}, P' = P + H| \; (C \boxminus R_1) + (C' \boxminus R_1) = i\}|.
\end{aligned}
$$

The second last equality comes from the fact

$$
(R_4 \boxplus t_3) + (K_5 + H) = ((R_4 + H) \boxplus t_3) + K_5,
$$

which can be easily verified by the computer simulation.

Note that condition (A) is a necessary condition for the following condition:

$$
\begin{aligned}
\Delta K &= (K_1, ..., K_8) + (K'_1, ..., K'_8) = (0, 0, 0, 0, 0, 0, 0, 0) \\
\Delta P &= P_1 + P'_{i'} = 0 \\
\Delta R &= (R_1, ..., R_8) + (R'_1, ..., R'_8) = (0, 0, 0, H, 0, 0, 0, 0),
\end{aligned}
$$

such that both of them produce the same differential sequence of $WD16$. However, we use condition (A) through the rest of the paper because it has an additional property that keeps the attacker informed once ΔR happens (see Section 5.2). $\square$

This theorem suggests that, after querying the encryption with every $P \in \mathbb{F}_2^{16}$ and obtaining the resultant output differentials, the attacker could have a *template sequence* $\Delta_{H,(K_5,K_6,K_7,K_8)}$ to search for parts of (K_5, K_6, K_7, K_8). The next theorem discloses the correspondence between $\Delta_{H,(K_5,K_6,K_7,K_8)}$ and (K_5, K_6, K_7, K_8).

Theorem 2. *Let $\Delta_{H,(K_5,K_6,K_7,K_8)}$ be obtained from Theorem 1. For $\kappa_5 \in \mathbb{F}_2^{16}$ and $\kappa_6 \in \mathbb{F}_2^{16}$, we have*

$$\Delta_{H,(K_5,K_6,K_7,K_8)} = \Delta_{H,(\kappa_5,\kappa_6,K_7,K_8)},$$

where K_6 and κ_6 belong to the same set $\Phi_i = \Phi_0 + i$, $0 < i \leq 15$, and Φ_0, of cardinality 2^{12}, is tabulated in Appendix C.

Proof: To prove, we discuss the correspondence between K_5, K_6, K_7, K_8 and the template sequence in a respective way.

Correspondence Between K_5 and DS: For the time being, let us consider $h(w, K) = f(f(w + K_5) + K_6)$ (a simplified $WD16$), where $f : \mathbb{F}_2^{16} \mapsto \mathbb{F}_2^{16}$ (as described in Section 2) is an injective function, we thus have, by letting $w = R_4 \boxplus t_3$ and $\theta = H$,

$$[h(w, K) + h(w + \theta, K)|w \in \mathbb{F}_2^{16}]$$
$$= [f(f(w + K_5) + K_6) + f(f(w + K_5 + \theta) + K_6)|w \in \mathbb{F}_2^{16}]$$
$$= [f(f(perm_1(w)) + K_6) + f(f(perm_1(w) + \theta) + K_6)|w \in \mathbb{F}_2^{16}]$$

It is clear from the context that $perm_1(w) = w + K_5$ is a permutation in $\mathbb{F}_2^m$, and, due to Property 4, $\Delta_{H,(K_5,K_6,K_7,K_8)}$ does not dependent on K_5.

Correspondence Between K_6 and DS: First of all, we define the following auxiliary variables for convenience:

- λ_i, $i = 1, ..., q$, are q possible output differences of f, given the input difference is θ.
- $\Gamma_i = \{f(w)|f(w) + f(w + \theta) = \lambda_i, w \in \mathbb{F}_2^{16}\}$, $i = 1, ..., q$, are q disjoint partitions of $\mathbb{F}_2^{16}$ such that: (1) Eq. (29) holds, otherwise there is a $w \in (\Gamma_i \cap \Gamma_j)$, $1 \leq i \neq j \leq q$, such that $f(w) + f(w + \theta)$ produces output differences λ_i and λ_j, $\lambda_i \neq \lambda_j$, which is impossible; (2) Eq. (30) holds, otherwise there is a $w \in (\mathbb{F}_2^{16} - \cup_{i=1}^q \Gamma_i)$, that produces an output difference $\notin \{\lambda_1, ..., \lambda_q\}$, which contradicts our definition.
- $\Phi_0 = \cap_{i=1}^q \{k|f(w) + k : \Gamma_i \mapsto \Gamma_i, k \in \mathbb{F}_2^{16}\}$. Intuitively, Φ_0 encompasses all possible keys, which make $f(w) + k$ a permutation in Γ_i, $i = 1, ..., q$.

Furthermore, let us consider two cases: (1) $K_6 \in \Phi_0$; and (2) $K_6 \in$ a coset of Φ_0.

For case (1), i.e., $K_6 \in \Phi_0$, the above equations can be further written as, by setting $perm_2(w) = f(perm_1(w)) + K_6$,

$$[f(f(perm_1(w)) + K_6) + f(f(perm_1(w) + \theta) + K_6)|w \in \mathbb{F}_2^{16}]$$
$$= [f(f(perm_1(w)) + K_6) + f(f(perm_1(w)) + \lambda_i + K_6)|w \in \Gamma_i] \quad \text{for } i = 1, ..., q$$
$$= [f(f(perm_1(w)) + K_6) + f(f(perm_1(w)) + K_6 + \lambda_i)|w \in \Gamma_i] \quad \text{for } i = 1, ..., q$$
$$= [f(perm_2(w)) + f(perm_2(w) + \lambda_i)|w \in \Gamma_i] \quad \text{for } i = 1, ..., q$$
$$= [f(w) + f(w + \lambda_i)|w \in \Gamma_i] \quad \text{for } i = 1, ..., q$$

The above equation holds because of Property 8, e.g., for $K_6 \in \Phi_0$, every $[f(w) + f(w + \lambda_i) | w \in \Gamma_i]$ produces the same sub-differential sequence. Therefore, the overall differential sequence stays the same for every $K_6 \in \Phi_0$. Stating in another way, providing K_6 and κ_6 are both in Φ_0, $\Delta_{H,(K_5,K_6)} = \Delta_{H,(\kappa_5,\kappa_6)}$.

The above derivation is further confirmed through extensive experiments, where we found

$$(\lambda_1, ..., \lambda_6) = (0x30cc, 0x6198, 0x9264, 0xa2a8, 0xc330, 0xf3fc),$$

$(\Gamma_1, ...\Gamma_6)$, and Φ_0 as tabulated in Appendix C, which is of cardinality 2^{12}.

In what follows, we prove case (2), i.e., the above equations are true for $K_6 \in \Phi_i = \Phi_0 + i$. This is because, by letting $K_6 = \triangleright K_6 + \triangleleft K_6$ such that $\triangleright K_6 \in \Phi_0$,

$$
\begin{aligned}
& [f(f(perm_1(w)) + K_6) + f(f(perm_1(w)) + K_6 + \lambda_i) | w \in \Gamma_i] && \text{for } i = 1, ..., q \\
= & [f(f(perm_1(w)) + \triangleright K_6 + \triangleleft K_6) + f(f(perm_1(w)) + \triangleright K_6 + \triangleleft K_6 + \lambda_i) | w \in \Gamma_i] && \text{for } i = 1, ..., q \\
= & [f(perm_3(w) + \triangleleft K_6) + f(perm_3(w) + \triangleleft K_6 + \lambda_i) | w \in \Gamma_i] && \text{for } i = 1, ..., q \\
= & [f(w + \triangleleft K_6) + f(w + \triangleleft K_6 + \lambda_i) | w \in \Gamma_i] && \text{for } i = 1, ..., q
\end{aligned}
$$

The second last equation holds because of our proof of case (1) (by letting $perm_3(w) = f(perm_1(w)) + \triangleright K_6$).

In addition, it is clear that:

- the sub-differential sequence $[f(w + \triangleleft K_6) + f(w + \triangleleft K_6 + \lambda_i) | w \in \Gamma_i]$, is different from $[f(w) + f(w + \lambda_i) | w \in \Gamma_i]$ as long as $\triangleleft K_6 \neq 0$. So is the overall differential sequence with overwhelming probability.
- for $K_6 = \triangleright K_6 + \triangleleft K_6$, $\kappa_6 = \triangleright \kappa_6 + \triangleleft \kappa_6$, $K_6 \neq \kappa_6$, the sub-differential sequence $[f(w + \triangleleft K_6) + f(w + \triangleleft K_6 + \lambda_i) | w \in \Gamma_i]$, is the same as $[f(w + \triangleleft \kappa_6) + f(w + \triangleleft \kappa_6 + \lambda_i) | w \in \Gamma_i]$ as long as $\triangleleft K_6 = \triangleleft \kappa_6$. This is due to the possibility that $\triangleright K_6, \triangleright \kappa_6 \in \Phi_0$, $\triangleright K_6 \neq \triangleright \kappa_6$ could yield $\triangleleft K_6 = \triangleleft \kappa_6$.

From the accusation above and our extensive experiments, it can be concluded that the key space of $K_6 \in \mathbb{F}_2^{16}$ has been divided into 16 cosets, i.e., $\Phi_0, ..., \Phi_{15}$, and each is of cardinality 2^{12}.

Correspondence Between (K_7, K_8) **and DS**: We carry on all the notations above for K_7 except setting $h(w, K) = f(f(f(f(w + K_5) + K_6) + K_7) + K_8)$. We found that, for K_7, Φ_0 is always a empty set because too many λ_i divides $\mathbb{F}_2^{16}$ into numerous tiny subspaces Γ_i, for which there is no K_7 could make $f(w) + K_7$ a permutation in every Γ_i, $i = 1, ..., q$. Same phenomenon happens to K_8. In all, each choice of (K_7, K_8) produces a different differential sequence, which is further confirmed empirically. $\square$

Attacking $WD16^{-1}$ **in Decryption**: Similar attack can be performed against the decryption. By assuming R_1 and R_1' are known and letting h in Definition 1 be the last invocation of $WD16^{-1}$, i.e., Eq. (28), we have the following results for our attack.

Theorem 3. *With the condition (B), the differential sequence of the last $WD16^{-1}$ in the decryption at $\theta = H$ can be extracted from executing the entire decryption.*

Proof: First of all, when condition (B) holds, we have,

$$
\begin{aligned}
u_3 &= WD16^{-1}(C \boxminus R_1, K_8, K_7, K_6, K_5) \\
&= WD16^{-1}((C+H) \boxminus R_1', (K_8+H), K_7', K_6', K_5') = u_3' \\
u_2 &= WD16^{-1}(u_3 \boxminus R_4, K_4 + R_8, K_3 + R_7, K_2 + R_6, K_1 + R_5) \\
&= WD16^{-1}(u_3' \boxminus R_4', (K_4+H)+(R_8+H), K_3'+R_7', K_2'+R_6', K_1'+R_5') = u_2' \\
u_1 &= WD16^{-1}(u_2 \boxminus R_3, K_8 + R_8, K_7 + R_7, K_6 + R_6, K_5 + R_5) \\
&= WD16^{-1}(u_2' \boxminus R_3', (K_8+H)+(R_8+H), K_7'+R_7', K_6'+R_6', K_5'+R_5') = u_1'
\end{aligned}
$$

Next, $\Delta_{H,(K_4,K_3,K_2,K_1)} = [z_0, z_1, ..., z_{2^{16}-1}]$ can be extracted, where,

$$
\begin{aligned}
z_i &= |\{u_1 \in \mathbb{F}_2^{16}|\ (WD16^{-1}(u_1 \boxminus R_2, K_4, K_3, K_2, K_1) + WD16^{-1}(u_1' \boxminus R_2', K_4', K_3', K_2', K_1')) = i\}| \\
&= |\{u_1 \in \mathbb{F}_2^{16}|\ (WD16^{-1}(u_1 \boxminus R_2, K_4, K_3, K_2, K_1) + WD16^{-1}(u_1' \boxminus R_2', K_4 + H, K_3, K_2, K_1)) = i\}| \\
&= |\{C \in \mathbb{F}_2^{16}, C' = C + H|\ (P \boxplus R_1) + (P' \boxplus R_1) = i\}|.
\end{aligned}
$$

$\square$

A similar theorem describes the correspondence between $\Delta_{H,(K_4,K_3,K_2,K_1)}$ and (K_4, K_3, K_2, K_1).

Theorem 4. *Let $\Delta_{H,(K_4,K_3,K_2,K_1)}$ be obtained from Theorem 3. For $\kappa_1 \in \mathbb{F}_2^{16}$ and $\kappa_4 \in \mathbb{F}_2^{16}$,*

$$
\Delta_{H,(K_4,K_3,K_2,K_1)} = \Delta_{H,(\kappa_4,K_3,K_2,\kappa_1)},
$$

where K_4 and κ_4 belong to the same set $\Phi_i = \Phi_0 + i$, $0 \le i \le 2^{12} - 1$, and $\Phi_0 = \{0x0000, 0x0010, ..., 0x00f0\}$.

Proof: Similar as Theorem 2, except that we could easily observe from the experimental data that $\Phi_0 = \{0x0000, 0x0010, 0x0020, ..., 0x00f0\}$. $\square$

Visualization of Differential Sequences From HB-2: We provide several examples of the differential sequences used in our experiments in Appendix B. Fig. 4 to Fig. 6 are the ones obtained from the last invocation of $WD16$ in the encryption with $IV = (0,0,0,0)$ and different keys randomly selected. Fig. 7 to Fig. 9 in Appendix B are the ones obtained from the last invocation of $WD16^{-1}$ in the decryption with $IV = (0,0,0,0)$ and different keys randomly selected. All of the sequences are substantially different from each other, which exhibits their correlations to the underlying keys in an intuitive way.

4.5 Local Search in DSA

After the template sequence has been captured, the attacker could, in an off-line environment, launches $h(w, K) = WD16$ ($h(w, K) = WD16^{-1}$ resp.) to search for parts of (K_5, K_6, K_7, K_8) ($(K_1, K_2, K_3, K_4$ resp.), which is called the *local search* in DSA. Through the local search, the attacker recovers 36-bit (44-bit resp.) information regarding the key.

A naive way to search locally is to produce a complete local differential sequence from $[h(w, K) + h(w + H, K) | w \in \mathbb{F}_2^{16}]$ with a random K at first, comparing each entry of which with the corresponding entry of the template sequence. The cost per key trial is 2^{16} executions of $h(w, K)$s and 2^{16} comparisons.

The efficiency of this method can be substantially improved if the early-abort strategy [30] is adapted, i.e., given the ith entry in the local differential sequence is greater than the ith entry in the template sequence, one could assert that the trial key is incorrect and terminate the search in advance. We present this improved local search algorithm below.

1: let TDS be the template sequence obtained
2: initiate the local differential sequence LDS as a list of 2^{16} "0"s
3: **for** w from 0 to $2^{16} - 1$ **do**
4:    randomly choose K
5:    $diff \leftarrow h(w + K) + h(w + H + K)$
6:    $LDS[diff] \leftarrow LDS[diff] + 1$
7:    **if** $LDS[diff] > TDS[diff]$ **then**
8:       return NULL
9:    **end if**
10: **end for**
11: **if** $LDS[w] = TDS[w]$ for $w = 0, 1, ..., 2^{16} - 1$ **then**
12:    return K
13: **end if**

The theoretical derivation of the time complexity of the above algorithm could be quite cumbersome. Instead, we recorded the number of the for-loops that are actually executed, denoted as l, during the search. Through repeated testings, we found that, in average, $1.640 < l < 1.660$ for-loops are spent per key trial for both local searches using $WD16(.)$ and $WD16^{-1}(.)$. Thus, we conclude the cost per key trial of our local search algorithm is 1.65 executions of (a pair of) $h(w, K)$s .

4.6 Differential Sequence Analysis (DSA) Against HB-2 and Its Time Complexity

We are ready to list out the steps performed by the attacker during the key recovery phase, as below.

1. When condition (A) holds, the attacker extracts the template sequence $\Delta_{H,(K_5,K_6,K_7,K_8)}$ using $((C \boxminus R_1) + (C' \boxminus R_1))$, where C and C' can be obtained by querying the encryption with P and $P' = P + H$, and R_1 and R'_1 are obtained in the preparation phase. Then, the attacker locally searches 36-bit of (K_5, K_6, K_7, K_8) using the proposed local search algorithm.
2. Similarly, utilizing the decryption, when condition (B) holds, the attacker extracts another template sequence $\Delta_{H,(K_4,K_3,K_2,K_1)}$ using $(P \boxplus R_1) + (P' \boxplus R_1)$, and guesses to determine 44-bit of (K_4, K_3, K_2, K_1) using the proposed local search algorithm.
3. After that, the attacker searches the remaining 48-bit of the key using 2^{48} trial encryptions[1].

The overall complexity of the above steps is

$$\underbrace{2^{36} \times 1.65}_{\text{determine 36-bit of } (K_5, ..., K_8)} + \underbrace{2^{44} \times 1.65}_{\text{determine 44-bit of } (K_1, ..., K_4)} + \underbrace{2^{48}}_{\text{determine the rest}} \approx 2^{48.14},$$

where negligible memory is required by each steps.

---

[1] In fact, $2^{48} + 2^{32} + 2^{16} = 2^{48.00002201}$ trial encryptions and three plaintext-ciphertext pairs are required.

5   A Probabilistic Realization of Conditions (A) and (B)

The attacks in the last section solely depends on the occurrences of conditions (A) and (B), to reach ΔRs of which seems unpractical at the first glance as the initialization of HB-2 makes the internal states unpredictable. In this section, we show a probabilistic approach to realize these conditions – when the internal states of two HB-2 instances are respectively random, there is a certain chance that the attacker could get the desired differentials in the internal states. To this end, we study how to randomize the internal states of HB-2 at first, and, how to determine whether the desired ΔRs happen.

5.1   Randomize the Internal States

There are two ways for the adversary to affect the internal states of HB-2:

- Providing the key is fixed, it is clear, from Eq. (1)-(12), that $(IV_1, ..., IV_4) \mapsto (R_1, ..., R_4)$ is an injective mapping and so is $(IV_1, ..., IV_4) \mapsto (R_5, ..., R_8)$. Therefore, the attacker could easily generate 2^{64} (out of 2^{128}) different internal states by choosing different IVs and launching the initialization.
- For a fixed key and a particular IV, the attacker could choose plaintext P_1 to feed HB-2 at first. If a state transition graph is drawn, we can see that the starting state, i.e., $R^{(1)}$, transits to 2^{16} neighboring states while each $P_1 \in \mathbb{F}_2^{16}$ is encrypted. Next, if another encryption is performed, e.g., encrypting P_2, each of these "neighboring states" again transits to another 2^{16} states providing P_2 takes every value in $\mathbb{F}_2^{16}$. By continuing this process, we would have all 2^{128} states covered in this graph. Therefore, to produce a set of random internal states, i.e., $\{R^{(1)}, R^{(2)}, ...\}$, we could, as shown in Fig. 3, feed the encryptions with a plaintext chain where P_i is selected uniformly at random in $\mathbb{F}_2^{16}$ for $i = 1, 2, ...$ Similarly, a ciphertext chain could be fed to the decryption oracle to generate a set of random internal states as well. Note that feeding HB-2 encryption with a chain of N random inputs is equivalent to perform an N-step 2^{16}-dimensional random walk in its state transition graph. Therefore, $|\{R^{(1)}, R^{(2)}, ...\}| \approx N$ if $N \ll 2^{128}$ [15].
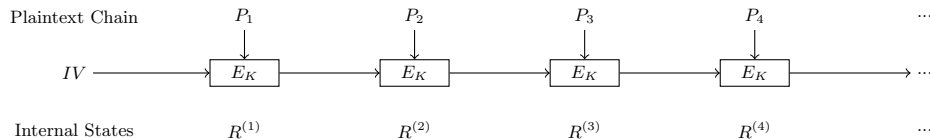
Fig. 3. Feeding HB-2 Encryption with a Plaintext Chain

Therefore, the algorithm below provides, to the later steps, the randomized internal states of two running HB-2 instances through an effort-saving way – one instance initializes a random IV and encrypts one random plaintext, while the other one, besides initializes a random IV, encrypts N random plaintexts consecutively. Since $\{R^{(1)}, R^{(2)}, ..., R^{(N)}\}$ is a set of random variables as analyzed, $\{R^{(1)} + R'^{(1)}, R^{(2)} + R'^{(1)}, ..., R^{(N)} + R'^{(1)}\}$ must also be a set of random variables.

1: Let $R^{(i)} \Leftarrow E(P_i, K)$ be the internal state $R^{(i)}$ after encrypting $P_1, ..., P_i$
2: Randomly choose IV' and P_1', $R'^{(1)} \Leftarrow E(P_1', K)$
3: Randomly choose IV
4: **for** i from 1 to N **do**
5:    Randomly choose P_i, $R^{(i)} \Leftarrow E(P_i, K)$
6:    **if** $R'^{(1)} + R^{(i)} = \Delta R$ **then**
7:       return "ΔR happens"
8:    **end if**
9: **end for**

Note that, currently, the given algorithm is only a skeleton for our attack, which is discussed in more detail in the next subsections and the full-fledged version is given at last. Nevertheless, we can already sense an interesting property from this skeleton algorithm.

Property 9. In the algorithm above, a certain ΔR happens with 0.5 probability when $N = 2^{64}$.

Proof. This property holds due to the birthday paradox. $\qquad\qquad\qquad\qquad\qquad\qquad\square$

5.2   Determine while Guessing

To inform the attacker during the attempting, as long as condition (A) (condition (B) resp.) happens, we use one unusual differential characteristic in the encryption (decryption resp.), as first pointed out by HB-2's designers, such that the differentials in the internal states, secret keys and the inputs can be maintained and entered into the next round, i.e., for a positive integer i,

$$(\Delta P_i, \Delta K, \Delta R^{(i)}) = (\Delta P_{i+1}, \Delta K, \Delta R^{(i+1)}).$$

Therefore, the following theorem holds.

Theorem 5. *Let $\Delta_{H,(K_5,K_6,K_7,K_8)}^{(i')}$ ($\Delta_{H,(K_4,K_3,K_2,K_1)}^{(i')}$ resp.) be the differential sequence produced by the two encryption instances (two decryption instances resp.) with internal states $R^{(1)}$ and $R'^{(i')}$ and let $\Delta_{H,(K_5,K_6,K_7,K_8)}^{(i'+1)}$ ($\Delta_{H,(K_4,K_3,K_2,K_1)}^{(i'+1)}$ resp.) be the differential sequence produced by the two encryption instances (two decryption instances resp.) with internal states $R^{(2)}$ and $R'^{(i'+1)}$ (call $\Delta_{H,K}^{(i')}$ and $\Delta_{H,K}^{(i'+1)}$ neighboring template sequences). Therefore,*

- *If condition (A) happens during encryption, the adversary observes two identical neighboring template sequences, i.e.,*
$$\Delta_{H,(K_5,K_6,K_7,K_8)}^{(i')} = \Delta_{H,(K_5,K_6,K_7,K_8)}^{(i'+1)};$$
otherwise, the above equation holds with negligible probability.
- *If condition (B) happens during decryption, the adversary observes two identical neighboring template sequences, i.e.,*
$$\Delta_{H,(K_4,K_3,K_2,K_1)}^{(i')} = \Delta_{H,(K_4,K_3,K_2,K_1)}^{(i'+1)};$$
otherwise, the above equation hold with negligible probability.

Proof: It follows from Definition 1 and Property 1. □

Therefore, the above theorem can serve as an algorithm to determine the occurrences of condition (A) or condition (B), i.e., it returns either $(Success, \Delta_{H,K}^{(i')}, R_1^{(1)}, R_1^{(2)})$ or $(False, NULL, NULL, NULL)$ to the key recovery phase. Unfortunately, in this algorithm, the correct template sequences can only be extracted with the correct $R_1^{(1)}$ and $R_1^{(2)}$ due to Theorem 1 and Theorem 3. For instance, using the encryption, the two neighboring sequences are

$$\Delta^{(i')} = (z_0^{(i')}, z_1^{(i')}, ..., z_{65535}^{(i')}) \tag{32}$$

$$\Delta^{(i'+1)} = (z_0^{(i'+1)}, z_1^{(i'+1)}, ..., z_{65535}^{(i'+1)}) \tag{33}$$

where

$$z_j^{(i')} = |\{P_1 \in \mathbb{F}_2^{16}, P'_{i'} = P_1 + H| \ (C_1 \boxminus R_1^{(1)}) + (C'_{i'} \boxminus R'^{(i')}_1) = j\}| \qquad \text{and}$$

$$z_j^{(i'+1)} = |\{P_2 \in \mathbb{F}_2^{16}, P'_{i'+1} = P_2 + H| \ (C_2 \boxminus R_1^{(2)}) + (C'_{i'+1} \boxminus R'^{(i'+1)}_1) = j\}|$$

Henceforth, it is true that by guessing $R_1^{(1)}$ and $R_1^{(2)}$, the algorithm above would cost 2^{32} encryptions/decryptions per execution.

To improve its efficiency, we make use of the *raw neighboring sequences* and their correlations, where the raw neighboring sequences are defined as below:

$$\underline{\Delta^{(i')}} = (\underline{z_0^{(i')}}, \underline{z_1^{(i')}}, ..., \underline{z_{65535}^{(i')}})$$

$$\underline{\Delta^{(i'+1)}} = (\underline{z_0^{(i'+1)}}, \underline{z_1^{(i'+1)}}, ..., \underline{z_{65535}^{(i'+1)}})$$

where

$$\underline{z_j^{(i')}} = |\{P_1 \in \mathbb{F}_2^{16}, P'_{i'} = P_1 + H| \ C_1 + C'_{i'} = j\}| \qquad \text{and}$$

$$\underline{z_j^{(i'+1)}} = |\{P_2 \in \mathbb{F}_2^{16}, P'_{i'+1} = P_2 + H| \ C_2 + C'_{i'+1} = j\}|.$$

In addition, we have the following hypothesis regarding the correlations of the raw neighboring sequences, due to the fact that the modulo addition is only first-order correlation-immune:

Hypothesis 1 *The correlation of the two raw neighboring sequences derived from the two identical neighboring sequences is stronger than that of the two raw neighboring sequences derived from the two distinct neighboring sequences.*

This hypothesis has been empirically verified, i.e., we found that, for the two identical neighboring sequences, the corresponding two raw neighboring sequences always have more than 30000 (out of 65536) identical entries, i.e.,

$$Corr(\underline{\Delta^{(i')}}, \underline{\Delta^{(i'+1)}}) = |\{\underline{z_j^{(i')}} = \underline{z_j^{(i'+1)}}, j = 0, 1, ..., 65535\}| > 30000, \text{ iff } \Delta^{(i')} = \Delta^{(i'+1)},$$

where $Corr(.,.)$ is the non-normalized correlation.

On the contrary, for the two distinct neighboring sequences, the corresponding two raw neighboring sequences always have less than 19000 (out of 65536) identical entries, i.e.,

$$Corr(\underline{\Delta^{(i')}}, \underline{\Delta^{(i'+1)}}) = |\{\underline{z_j^{(i')}} = \underline{z_j^{(i'+1)}}, j = 0, 1, ..., 65535\}| < 19000, \text{ iff } \Delta^{(i')} \neq \Delta^{(i'+1)}.$$

By treating the correlation of the raw neighboring sequences as a criterion, Theorem 5 is now able to return whether $\Delta^{(i')}$ equals $\Delta^{(i'+1)}$ within 2^{16} time complexity. Once the identical neighboring sequences are identified, the adversary is able to guess to recover $R_1^{(1)}$ and $R_1^{(2)}$ with 2^{32} effort in time.

5.3 Preparation Phase and Its Time Complexity

We recap the whole process in the preparation phase for the encryption as shown below, which is an extension of the skeleton algorithm we shown before. Note that the preparation using the decryption is similar and omitted here.

1: randomly choose IV' and P_1', $R'^{(1)} \Leftarrow E(P_1', K)$
2: randomly choose IV
3: randomly choose a constant P_2'
4: **for** i from 1 to $N = 2^{64}$ **do**
5:     randomly choose P_i, $R^{(i)} \Leftarrow E(P_i, K)$
6:     generate $\Delta^{(i)}$ using $R'^{(1)}$ and $R^{(i)}$
7:     $R'^{(2)} \Leftarrow E(P_2', K)$
8:     $R^{(i+1)} \Leftarrow E(P_{i+1}, K)$ where $P_{i+1} = P_2' + H$
9:     generate $\Delta^{(i+1)}$ using $R'^{(2)}$ and $R^{(i+1)}$
10:     **if** $Corr(\Delta^{(i)}, \Delta^{(i+1)}) > 30000$ **then**
11:        guess to determine $R_1^{(1)}$ and $R_1^{(2)}$
12:        recover $\Delta_{H,K}^{(i')}$ from the raw neighboring sequences
13:        return $(Success, \Delta_{H,K}^{(i')}, R_1^{(1)}, R_1^{(2)})$, keep current states and enter the key recovery phase
14:     **end if**
15:     decrypt using C_2' and C_{i+1} to roll back HB-2's states to $R'^{(1)}$ and $R^{(i)}$
16: **end for**
17: return $(False, NULL, NULL, NULL)$

Using the encryption (decryption resp.) only, the attacker has 0.5 probability to reach condition (A) (condition (B) resp.) with $2^{64} \times 2^{16} = 2^{80}$ time complexity. After that, he is able to guess to determine $R_1^{(1)}$ and $R_1^{(2)}$ with additional 2^{32} effort in time. In all, the time complexity of the preparation phase is

$$\underbrace{2^{64} \times 2^{16}}_{\text{test whether the condition happens}} + \underbrace{2^{32}}_{\text{guess to determine } R_1 \text{s}} + \underbrace{2^{16}}_{\text{recover the template seuqnece}} \approx 2^{80}.$$

It is worth mentioning that to succeed with probability 1, the preparation phase requires $2^{128+16} = 2^{144}$ effort in time, which is slower than the exhaustive search.

6 Concluding Remarks

In this paper, we present a novel cryptanalytic technique called differential sequence analysis (DSA), which is especially effective if the differential sequence reflecting parts of a cipher associated with

parts of the key can be obtained. In addition, we demonstrate the application of this technique, that constitutes the key recovery of the lightweight block cipher Hummingbird-2 with $2^{48.14}$ time complexity, given particular conditions hold in its internal states, secret keys and the inputs. Furthermore, we investigate how to reach these conditions in our preparation phase with 0.5 chance and 2^{80} effort in time. To the best of our knowledge, this is the first cryptanalytic result of the full Hummingbird-2.

The attack presented against Hummingbird-2 is a special case of the general DSA, to build the theoretic framework of which is part of our future work. In addition, it will be evaluated in the recent future: (1) whether the generalized DSA provides even better results against Hummingbird-2 and other potentially vulnerable ciphers, especially the ones with small block size and with internal states, e.g., stateful block ciphers [24]; (2) the possibility that the generalized DSA can work with other cryptanalysis technologies, e.g., meet-in-the-middle.

Acknowledgement

References

1. J.P. Aumasson, M. Naya-Plasencia and M.J.O. Saarinen, Practical attack on 8 rounds of the lightweight block cipher KLEIN, to appear in Proceedings of *INDOCRYPT'11*, pp.1–13, 2011.
2. M. Abdelraheem, G. Leander and E. Zenner, Differential cryptanalysis of round-reduced PRINTcipher: computing roots of permutations, *Fast Software Encryption, FSE'11*, LNCS 6733, pp. 1–17, 2011.
3. M. Ågren, Some instant-and practical-time related-key attacks on KTANTAN32/48/64, *Selected Areas in Cryptography, SAC'11*, LNCS 7118, pp. 213–229, 2012.
4. A. Bogdanov and C. Rechberger, A 3-subset meet-in-the-middle attack: cryptanalysis of the lightweight block cipher KTANTAN, *Selected Areas in Cryptography*, LNCS 6544, pp. 229–240, 2011.
5. E. Biham, A. Biryukov and A. Shamir, Cryptanalysis of Skipjack reduced to 31 rounds using impossible differentials, *Journal Of Cryptology*, vol. 18, no. 4, pp. 291–311, 1999.
6. C. Blondeau and B. Gérard, Multiple differential cryptanalysis: theory and practice, *Fast Software Encryption, FSE'11*, LNCS 6733, pp. 35-54, 2011.
7. A. Bogdanov and C. Rechberger, A 3-subset meet-in-the-middle attack: cryptanalysis of the lightweight block cipher KTANTAN, *Selected Areas in Cryptography, SAC'10*, LNCS 6544, pp. 229–240, 2010.
8. A. Biryukov and A. Shamir, Structural cryptanalysis of SASAS, *Journal of cryptology*, vol. 23, no. 4, pp. 505–518, 2010.
9. A. Biryukov, I. Kizhvatov and B. Zhang, Cryptanalysis of the Atmel cipher in SecureMemory, CryptoMemory and CryptoRF, *Applied Cryptography and Network Security, ACNS'11*, LNCS 6715, pp. 91–109, 2011.
10. Q. Chai, Design and analysis of security schemes for low-cost RFID systems, *PhD Thesis, University of Waterloo*, 2012.
11. J. Cho, Linear cryptanalysis of reduced-round PRESENT, *The Cryptographers' Track at the RSA Conference, CT-RSA'10*, LNCS 5985, pp. 302–317, 2010.
12. Q. Chai, X. Fan and G.Gong, An ultra-efficient key recovery attack on the lightweight stream cipher A2U2, *Cryptology ePrint Archive: Report 2011/247*, pp. 1–4, 2011.
13. B. Collard and F.X. Standaert, A statistical saturation attack against the block cipher PRESENT, *The Cryptographers' Track at the RSA Conference, CT-RSA'09*, LNCS 5473, pp. 195–210, 2009.
14. C. De Canniere, O. Dunkelman and M. Knežević, KATAN and KTANTAN – a family of small and efficient hardware-oriented block ciphers, *Cryptographic Hardware and Embedded Systems, CHES'09*, LNCS 5747, pp. 272–288, 2009.

15. A. Dvoretzky and P. Erdos, Some problems on random walk in space, *In Proceedings of Second Berkeley Symposium on Mathematical Statistics and Probability*, vol. 353, pp. 353–367, 1951.
16. D. Engels, X. Fan, G. Gong, H. Hu and E. Smith, Hummingbird: ultra-lightweight cryptography for resource-constrained devices, *Financial Cryptography and Data Security, FC'10*, pp. 3–18, 2010.
17. D. Engels, M.J.O. Saarinen and E. Smith, The Hummingbird-2 lightweight authenticated encryption algorithm, *Workshop on RFID Security, RFIDSec'11*, LNCS 7055, pp. 19–31, 2011.
18. X. Fan and G. Gong, On the security of Hummingbird-2 against side-channel Cube attacks, *Western European Workshop on Research in Cryptology, WEWoRC'11*, pp. 100–104, 2011.
19. M. Feldhofer, J. Wolkerstorfer and V. Rijmen, AES implementation on a grain of sand, *Information Security, IEE Proceedings*, vol. 152, no. 1, pp. 13-20, 2005.
20. Z. Gong, S. Nikova and Y.W. Law, Klein: a new family of lightweight block ciphers, *Workshop on RFID Security, RFIDSec'11*, LNCS 7055, pp.1–18, 2011.
21. J. Guo, T. Peyrin, A. Poschmann and M.Robshaw, The LED block cipher, *Cryptographic Hardware and Embedded Systems, CHES'11*, LNCS 6917, pp. 326–341, 2011.
22. F.D. Garcia, P. van Rossum, R. Verdult and R.W. Schreur, Dismantling SecureMemory, CryptoMemory and CryptoRF. *In Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS'10*, pp. 250-259, 2010.
23. K. Hwang, W. Lee, S. Lee, S. Lee and J. Lim, Saturation attacks on reduced round Skipjack, *Fast Software Encryption, FSE'02*, pp. 15–23, 2002.
24. A. Kiayias and M. Yung, cryptographic hardness based on the decoding of Reed-Solomon codes, *Automata, Languages and Programming*, pp. 783–783, 2002.
25. L. Knudsen, Truncated and higher order differentials, *Fast Software Encryption. FSE'95*, LNCS 1008, pp. 196–211, 1995.
26. L. Knudsen, G. Leander, A. Poschmann and M. Robshaw, PRINTcipher: a block cipher for IC-printing, *Cryptographic Hardware and Embedded Systems, CHES'10*, LNCS 6225, pp.16–32, 2011.
27. X. Lai, Higher order derivatives and differential cryptanalysis, *Kluwer International Series in Engineering and Computer Science*, pp. 227–227, 1994.
28. S. Lucks, The saturation attack: a bait for Twofish, *Fast Software Encryption, FSE'02*, pp. 187–205, 2002.
29. G. Leander, M.A. Abdelraheem, H. AlKhzaimi and E. Zenner, A cryptanalysis of PRINTcipher: the invariant subspace attack. *Advances in Cryptology, CRYPTO'11*, LNCS 6841, pp. 206-221, 2011.
30. J. Lu, J. Kim, N. Keller and O. Dunkelman, Improving the efficiency of impossible differential cryptanalysis of reduced Camellia and MISTY1, *The Cryptographers' Track at the RSA Conference, CT-RSA'08*, LNCS 4964, pp. 370–386, 2008.
31. G. Leander and A. Poschmann On the classification of 4 bit s-boxes, *Arithmetic of Finite Fields*, LNCS 4547, pp. 159-176, 2007.
32. K. Nohl, D. Evans, S. Starbug and H. Plötz, Reverse-engineering a cryptographic RFID tag, *In Proceedings of the 17th conference on Security symposium, USENIX'08*, pp. 185–193, 2008.
33. N. Courtois, G. Bard and D. Wagner, Algebraic and slide attacks on KeeLoq, *Fast Software Encryption, FSE'08*, LNCS 5086, pp. 97-115, 2008.
34. A. Poschmann, S. Ling and H. Wang, 256 bit standardized crypto for 650 GE–GOST revisited, *Cryptographic Hardware and Embedded Systems, CHES'10*, LNCS 6225, pp. 219–233, 2011.
35. C. Rolfes, A. Poschmann, G. Leander and C. Paar, Ultra-lightweight implementations for smart devices–security for 1000 gate equivalents. *In Proceedings of the 8th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications, CARDIS'08*, LNCS 5189, pp. 89–103, 2008.
36. M.J.O. Saarinen, Cryptanalysis of Hummingbird-1, *Fast Software Encryption, FSE'11*, LNCS 6733, pp. 328–341, 2011.
37. A. Shamir and E. Biham, Differential cryptanalysis of DES-like cryptosystems, *Advances in Cryptology, CRYPTO'90*, LNCS 537, pp. 2-21, 1990.
38. K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita and T. Shirai, Piccolo: an ultra-lightweight blockcipher, *Cryptographic Hardware and Embedded Systems, CHES'11*, LNCS 6917, pp. 342–357, 2011.

A   Side-channel Injection Attack to Recover (R_5, R_6, R_7, R_8)

As can be seen from Eq. (6)-(9), R_5, R_6, R_7, R_8 do not participate in the randomization process but simply record (by Xoring) the historical statuses of R_1, R_2, R_3, R_4 respectively. Therefore, following steps allow a side-channel attacker, who is able to inject "1" to a certain bit of the register storing R_j, $5 \leq j \leq 8$, to recover (R_5, R_6, R_7, R_8):

1. The attacker encrypts with a known IV and the target key to get a plaintext/cipher pair (P, C), where $P \in \mathbb{F}_2^{16}, C \in \mathbb{F}_2^{16}$;
2. He resets HB-2 and initializes HB-2 with the same IV and key. At any time during this initialization, he injects "1" to the qth bit, $0 \leq q \leq 15$, of the register which stores R_5. He then encrypts P and gets C'. If $C = C'$ (which implies the injection does not change the internal states of HB-2), the attacker in fact learns that the qth bit of R_5 is 1; otherwise it is 0. He repeats this step for every q in $\{0, 1, ..., 15\}$ to recover R_5;
3. Step (2) can be repeated to recover R_6, R_7 and R_8;

The cost of this injection attack to recover (R_5, R_6, R_7, R_8) is 64 injections and 64 invocations of HB-2 encryption. In addition, since the attacker has a large time window to perform the injection to the qth bit of R_j (any time during the rth iteration of the initialization), this side-channel attack seems quite practical.

B   Visualization of Differential Sequences

Fig. 4 to Fig. 6 are the ones obtained from the last invocation of $WD16$ in the encryption with $IV = (0,0,0,0)$ and different keys randomly selected. Fig. 7 to Fig. 9 are the ones obtained from the last invocation of $WD16^{-1}$ in the decryption with $IV = (0,0,0,0)$ and different keys randomly selected. All of the sequences looks substantially different from each other, which exhibits their correlations to the underlying keys in an intuitive way.
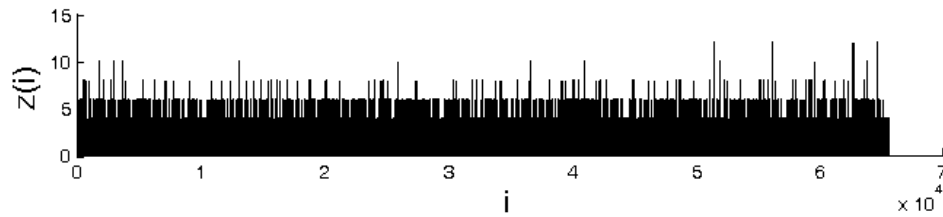
Fig. 4. DS from Enc. using $(K_5, K_6, K_7, K_8) = (\text{0xf1e3,0x524a,0xb28a,0xc987})$
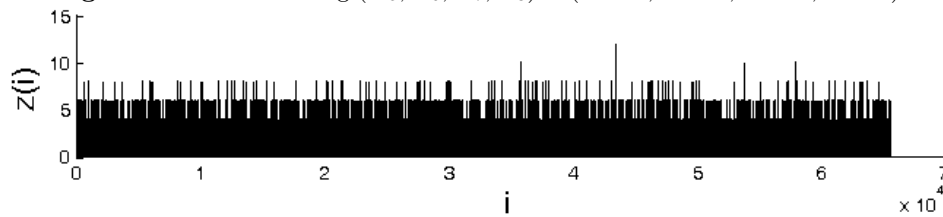
Fig. 5. DS from Enc. using $(K_5, K_6, K_7, K_8) = (\text{0x7c9f,0x0784,0x1c96,0xbcb4})$
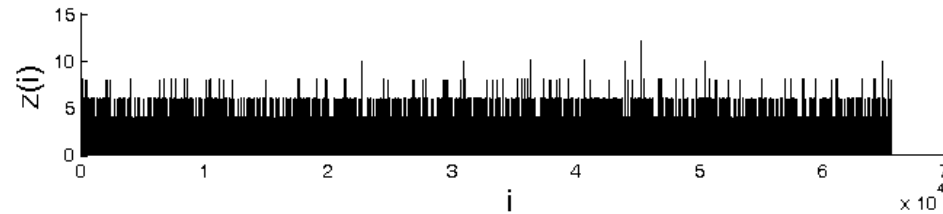
Fig. 6. DS from Enc. using $(K_5, K_6, K_7, K_8) = (\text{0x6b03,0xcf0c,0x1ba2,0xdc27})$
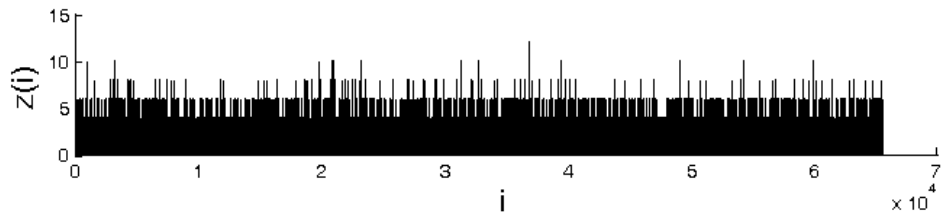
Fig. 7. DS from Dec. using $(K_1, K_2, K_3, K_4) = (0x5d67, 0xd0ef, 0x8cec, 0xa33a)$

Fig. 8. DS from Dec. using $(K_1, K_2, K_3, K_4) = (0x6601, 0x0bd8, 0xa6fa, 0xcede)$

Fig. 9. DS from Dec. using $(K_1, K_2, K_3, K_4) = (0x28dc, 0xbde1, 0x6e3d, 0xa56d)$

C   The Set Φ_0

0x0	0x10	0x20	0x30	0x40	0x50	0x60	0x70	0x80	0x90	0xa0	0xb0	0xc0	0xd0	0xe0	0xf0	0x105	0x115	0x125	0x135	0x145
0x155	0x165	0x175	0x185	0x195	0x1a5	0x1b5	0x1c5	0x1d5	0x1e5	0x1f5	0x20a	0x21a	0x22a	0x23a	0x24a	0x25a	0x26a	0x27a	0x28a	0x29a
0x2aa	0x2ba	0x2ca	0x2da	0x2ea	0x2fa	0x30f	0x31f	0x32f	0x33f	0x34f	0x35f	0x36f	0x37f	0x38f	0x39f	0x3af	0x3bf	0x3cf	0x3df	0x3ef
0x3ff	0x401	0x411	0x421	0x431	0x441	0x451	0x461	0x471	0x481	0x491	0x4a1	0x4b1	0x4c1	0x4d1	0x4e1	0x4f1	0x504	0x514	0x524	0x534
0x544	0x554	0x564	0x574	0x584	0x594	0x5a4	0x5b4	0x5c4	0x5d4	0x5e4	0x5f4	0x60b	0x61b	0x62b	0x63b	0x64b	0x65b	0x66b	0x67b	0x68b
0x69b	0x6ab	0x6bb	0x6cb	0x6db	0x6eb	0x6fb	0x70e	0x71e	0x72e	0x73e	0x74e	0x75e	0x76e	0x77e	0x78e	0x79e	0x7ae	0x7be	0x7ce	0x7de
0x7ee	0x7fe	0x802	0x812	0x822	0x832	0x842	0x852	0x862	0x872	0x882	0x892	0x8a2	0x8b2	0x8c2	0x8d2	0x8e2	0x8f2	0x907	0x917	0x927
0x937	0x947	0x957	0x967	0x977	0x987	0x997	0x9a7	0x9b7	0x9c7	0x9d7	0x9e7	0x9f7	0xa08	0xa18	0xa28	0xa38	0xa48	0xa58	0xa68	0xa78
0xa88	0xa98	0xaa8	0xab8	0xac8	0xad8	0xae8	0xaf8	0xb0d	0xb1d	0xb2d	0xb3d	0xb4d	0xb5d	0xb6d	0xb7d	0xb8d	0xb9d	0xbad	0xbbd	0xbcd
0xbdd	0xbed	0xbfd	0xc03	0xc13	0xc23	0xc33	0xc43	0xc53	0xc63	0xc73	0xc83	0xc93	0xca3	0xcb3	0xcc3	0xcd3	0xce3	0xcf3	0xd06	0xd16
0xd26	0xd36	0xd46	0xd56	0xd66	0xd76	0xd86	0xd96	0xda6	0xdb6	0xdc6	0xdd6	0xde6	0xdf6	0xe09	0xe19	0xe29	0xe39	0xe49	0xe59	0xe69
0xe79	0xe89	0xe99	0xea9	0xeb9	0xec9	0xed9	0xee9	0xef9	0xf0c	0xf1c	0xf2c	0xf3c	0xf4c	0xf5c	0xf6c	0xf7c	0xf8c	0xf9c	0xfac	0xfbc
0xfcc	0xfdc	0xfec	0xffc	0x1001	0x1011	0x1021	0x1031	0x1041	0x1051	0x1061	0x1071	0x1081	0x1091	0x10a1	0x10b1	0x10c1	0x10d1	0x10e1	0x10f1	0x1104
0x1114	0x1124	0x1134	0x1144	0x1154	0x1164	0x1174	0x1184	0x1194	0x11a4	0x11b4	0x11c4	0x11d4	0x11e4	0x11f4	0x120b	0x121b	0x122b	0x123b	0x124b	0x125b
0x126b	0x127b	0x128b	0x129b	0x12ab	0x12bb	0x12cb	0x12db	0x12eb	0x12fb	0x130e	0x131e	0x132e	0x133e	0x134e	0x135e	0x136e	0x137e	0x138e	0x139e	0x13ae
0x13be	0x13ce	0x13de	0x13ee	0x13fe	0x1400	0x1410	0x1420	0x1430	0x1440	0x1450	0x1460	0x1470	0x1480	0x1490	0x14a0	0x14b0	0x14c0	0x14d0	0x14e0	0x14f0
0x1505	0x1515	0x1525	0x1535	0x1545	0x1555	0x1565	0x1575	0x1585	0x1595	0x15a5	0x15b5	0x15c5	0x15d5	0x15e5	0x15f5	0x160a	0x161a	0x162a	0x163a	0x164a
0x165a	0x166a	0x167a	0x168a	0x169a	0x16aa	0x16ba	0x16ca	0x16da	0x16ea	0x16fa	0x170f	0x171f	0x172f	0x173f	0x174f	0x175f	0x176f	0x177f	0x178f	0x179f
0x17af	0x17bf	0x17cf	0x17df	0x17ef	0x17ff	0x1803	0x1813	0x1823	0x1833	0x1843	0x1853	0x1863	0x1873	0x1883	0x1893	0x18a3	0x18b3	0x18c3	0x18d3	0x18e3
0x18f3	0x1906	0x1916	0x1926	0x1936	0x1946	0x1956	0x1966	0x1976	0x1986	0x1996	0x19a6	0x19b6	0x19c6	0x19d6	0x19e6	0x19f6	0x1a09	0x1a19	0x1a29	0x1a39
0x1a49	0x1a59	0x1a69	0x1a79	0x1a89	0x1a99	0x1aa9	0x1ab9	0x1ac9	0x1ad9	0x1ae9	0x1af9	0x1b0c	0x1b1c	0x1b2c	0x1b3c	0x1b4c	0x1b5c	0x1b6c	0x1b7c	0x1b8c
0x1b9c	0x1bac	0x1bbc	0x1bcc	0x1bdc	0x1bec	0x1bfc	0x1c02	0x1c12	0x1c22	0x1c32	0x1c42	0x1c52	0x1c62	0x1c72	0x1c82	0x1c92	0x1ca2	0x1cb2	0x1cc2	0x1cd2
0x1ce2	0x1cf2	0x1d07	0x1d17	0x1d27	0x1d37	0x1d47	0x1d57	0x1d67	0x1d77	0x1d87	0x1d97	0x1da7	0x1db7	0x1dc7	0x1dd7	0x1de7	0x1df7	0x1e08	0x1e18	0x1e28
0x1e38	0x1e48	0x1e58	0x1e68	0x1e78	0x1e88	0x1e98	0x1ea8	0x1eb8	0x1ec8	0x1ed8	0x1ee8	0x1ef8	0x1f0d	0x1f1d	0x1f2d	0x1f3d	0x1f4d	0x1f5d	0x1f6d	0x1f7d
0x1f8d	0x1f9d	0x1fad	0x1fbd	0x1fcd	0x1fdd	0x1fed	0x1ffd	0x2002	0x2012	0x2022	0x2032	0x2042	0x2052	0x2062	0x2072	0x2082	0x2092	0x20a2	0x20b2	0x20c2
0x20d2	0x20e2	0x20f2	0x2107	0x2117	0x2127	0x2137	0x2147	0x2157	0x2167	0x2177	0x2187	0x2197	0x21a7	0x21b7	0x21c7	0x21d7	0x21e7	0x21f7	0x2208	0x2218
0x2228	0x2238	0x2248	0x2258	0x2268	0x2278	0x2288	0x2298	0x22a8	0x22b8	0x22c8	0x22d8	0x22e8	0x22f8	0x230d	0x231d	0x232d	0x233d	0x234d	0x235d	0x236d
0x237d	0x238d	0x239d	0x23ad	0x23bd	0x23cd	0x23dd	0x23ed	0x23fd	0x2403	0x2413	0x2423	0x2433	0x2443	0x2453	0x2463	0x2473	0x2483	0x2493	0x24a3	0x24b3
0x24c3	0x24d3	0x24e3	0x24f3	0x2506	0x2516	0x2526	0x2536	0x2546	0x2556	0x2566	0x2576	0x2586	0x2596	0x25a6	0x25b6	0x25c6	0x25d6	0x25e6	0x25f6	0x2609
0x2619	0x2629	0x2639	0x2649	0x2659	0x2669	0x2679	0x2689	0x2699	0x26a9	0x26b9	0x26c9	0x26d9	0x26e9	0x26f9	0x270c	0x271c	0x272c	0x273c	0x274c	0x275c
0x276c	0x277c	0x278c	0x279c	0x27ac	0x27bc	0x27cc	0x27dc	0x27ec	0x27fc	0x2800	0x2810	0x2820	0x2830	0x2840	0x2850	0x2860	0x2870	0x2880	0x2890	0x28a0
0x28b0	0x28c0	0x28d0	0x28e0	0x28f0	0x2905	0x2915	0x2925	0x2935	0x2945	0x2955	0x2965	0x2975	0x2985	0x2995	0x29a5	0x29b5	0x29c5	0x29d5	0x29e5	0x29f5
0x2a0a	0x2a1a	0x2a2a	0x2a3a	0x2a4a	0x2a5a	0x2a6a	0x2a7a	0x2a8a	0x2a9a	0x2aaa	0x2aba	0x2aca	0x2ada	0x2aea	0x2afa	0x2b0f	0x2b1f	0x2b2f	0x2b3f	0x2b4f
0x2b5f	0x2b6f	0x2b7f	0x2b8f	0x2b9f	0x2baf	0x2bbf	0x2bcf	0x2bdf	0x2bef	0x2bff	0x2c01	0x2c11	0x2c21	0x2c31	0x2c41	0x2c51	0x2c61	0x2c71	0x2c81	0x2c91
0x2ca1	0x2cb1	0x2cc1	0x2cd1	0x2ce1	0x2cf1	0x2d04	0x2d14	0x2d24	0x2d34	0x2d44	0x2d54	0x2d64	0x2d74	0x2d84	0x2d94	0x2da4	0x2db4	0x2dc4	0x2dd4	0x2de4
0x2df4	0x2e0b	0x2e1b	0x2e2b	0x2e3b	0x2e4b	0x2e5b	0x2e6b	0x2e7b	0x2e8b	0x2e9b	0x2eab	0x2ebb	0x2ecb	0x2edb	0x2eeb	0x2efb	0x2f0e	0x2f1e	0x2f2e	0x2f3e
0x2f4e	0x2f5e	0x2f6e	0x2f7e	0x2f8e	0x2f9e	0x2fae	0x2fce	0x2fde	0x2fee	0x2ffe	0x3003	0x3013	0x3023	0x3033	0x3043	0x3053	0x3063	0x3073	0x3083	
0x3093	0x30a3	0x30b3	0x30c3	0x30d3	0x30e3	0x30f3	0x3106	0x3116	0x3126	0x3136	0x3146	0x3156	0x3166	0x3176	0x3186	0x3196	0x31a6	0x31b6	0x31c6	0x31d6
0x31e6	0x31f6	0x3209	0x3219	0x3229	0x3239	0x3249	0x3259	0x3269	0x3279	0x3289	0x3299	0x32a9	0x32b9	0x32c9	0x32d9	0x32e9	0x32f9	0x330c	0x331c	0x332c
0x333c	0x334c	0x335c	0x336c	0x337c	0x338c	0x339c	0x33ac	0x33bc	0x33cc	0x33dc	0x33ec	0x33fc	0x3402	0x3412	0x3422	0x3432	0x3442	0x3452	0x3462	0x3472
0x3482	0x3492	0x34a2	0x34b2	0x34c2	0x34d2	0x34e2	0x34f2	0x3507	0x3517	0x3527	0x3537	0x3547	0x3557	0x3567	0x3577	0x3587	0x3597	0x35a7	0x35b7	0x35c7
0x35d7	0x35e7	0x35f7	0x3608	0x3618	0x3628	0x3638	0x3648	0x3658	0x3668	0x3678	0x3688	0x3698	0x36a8	0x36b8	0x36c8	0x36d8	0x36e8	0x36f8	0x370d	0x371d
0x372d	0x373d	0x374d	0x375d	0x376d	0x377d	0x378d	0x379d	0x37ad	0x37bd	0x37cd	0x37dd	0x37ed	0x37fd	0x3801	0x3811	0x3821	0x3831	0x3841	0x3851	0x3861
0x3871	0x3881	0x3891	0x38a1	0x38b1	0x38c1	0x38d1	0x38e1	0x38f1	0x3904	0x3914	0x3924	0x3934	0x3944	0x3954	0x3964	0x3974	0x3984	0x3994	0x39a4	0x39b4
0x39c4	0x39d4	0x39e4	0x39f4	0x3a0b	0x3a1b	0x3a2b	0x3a3b	0x3a4b	0x3a5b	0x3a6b	0x3a7b	0x3a8b	0x3a9b	0x3aab	0x3abb	0x3acb	0x3adb	0x3aeb	0x3afb	0x3b0e
0x3b1e	0x3b2e	0x3b3e	0x3b4e	0x3b5e	0x3b6e	0x3b7e	0x3b8e	0x3b9e	0x3bae	0x3bbe	0x3bce	0x3bde	0x3bee	0x3bfe	0x3c00	0x3c10	0x3c20	0x3c30	0x3c40	0x3c50
0x3c60	0x3c70	0x3c80	0x3c90	0x3ca0	0x3cb0	0x3cc0	0x3cd0	0x3ce0	0x3cf0	0x3d05	0x3d15	0x3d25	0x3d35	0x3d45	0x3d55	0x3d65	0x3d75	0x3d85	0x3d95	0x3da5
0x3db5	0x3dc5	0x3dd5	0x3de5	0x3df5	0x3e0a	0x3e1a	0x3e2a	0x3e3a	0x3e4a	0x3e5a	0x3e6a	0x3e7a	0x3e8a	0x3e9a	0x3eaa	0x3eba	0x3eca	0x3eda	0x3eea	0x3efa
0x3f0f	0x3f1f	0x3f2f	0x3f3f	0x3f4f	0x3f5f	0x3f6f	0x3f7f	0x3f8f	0x3f9f	0x3faf	0x3fbf	0x3fcf	0x3fdf	0x3fef	0x3fff	0x4001	0x4011	0x4021	0x4031	0x4041
0x4051	0x4061	0x4071	0x4081	0x4091	0x40a1	0x40b1	0x40c1	0x40d1	0x40e1	0x40f1	0x4104	0x4114	0x4124	0x4134	0x4144	0x4154	0x4164	0x4174	0x4184	0x4194
0x41a4	0x41b4	0x41c4	0x41d4	0x41e4	0x41f4	0x420b	0x421b	0x422b	0x423b	0x424b	0x425b	0x426b	0x427b	0x428b	0x429b	0x42ab	0x42bb	0x42cb	0x42db	0x42eb
0x42fb	0x430e	0x431e	0x432e	0x433e	0x434e	0x435e	0x436e	0x437e	0x438e	0x439e	0x43ae	0x43be	0x43ce	0x43de	0x43ee	0x43fe	0x4400	0x4410	0x4420	0x4430
0x4440	0x4450	0x4460	0x4470	0x4480	0x4490	0x44a0	0x44b0	0x44c0	0x44d0	0x44e0	0x44f0	0x4505	0x4515	0x4525	0x4535	0x4545	0x4555	0x4565	0x4575	0x4585
0x4595	0x45a5	0x45b5	0x45c5	0x45d5	0x45e5	0x45f5	0x460a	0x461a	0x462a	0x463a	0x464a	0x465a	0x466a	0x467a	0x468a	0x469a	0x46aa	0x46ba	0x46ca	0x46da
0x46ea	0x46fa	0x470f	0x471f	0x472f	0x473f	0x474f	0x475f	0x476f	0x477f	0x478f	0x479f	0x47af	0x47bf	0x47cf	0x47df	0x47ef	0x47ff	0x4803	0x4813	0x4823
0x4833	0x4843	0x4853	0x4863	0x4873	0x4883	0x4893	0x48a3	0x48b3	0x48c3	0x48d3	0x48e3	0x48f3	0x4906	0x4916	0x4926	0x4936	0x4946	0x4956	0x4966	0x4976
0x4986	0x4996	0x49a6	0x49b6	0x49c6	0x49d6	0x49e6	0x49f6	0x4a09	0x4a19	0x4a29	0x4a39	0x4a49	0x4a59	0x4a69	0x4a79	0x4a89	0x4a99	0x4aa9	0x4ab9	0x4ac9
0x4ad9	0x4ae9	0x4af9	0x4b0c	0x4b1c	0x4b2c	0x4b3c	0x4b4c	0x4b5c	0x4b6c	0x4b7c	0x4b8c	0x4b9c	0x4bac	0x4bbc	0x4bcc	0x4bdc	0x4bec	0x4bfc	0x4c02	0x4c12
0x4c22	0x4c32	0x4c42	0x4c52	0x4c62	0x4c72	0x4c82	0x4c92	0x4ca2	0x4cb2	0x4cc2	0x4cd2	0x4ce2	0x4cf2	0x4d07	0x4d17	0x4d27	0x4d37	0x4d47	0x4d57	0x4d67
0x4d77	0x4d87	0x4d97	0x4da7	0x4db7	0x4dc7	0x4dd7	0x4de7	0x4df7	0x4e08	0x4e18	0x4e28	0x4e38	0x4e48	0x4e58	0x4e68	0x4e78	0x4e88	0x4e98	0x4ea8	0x4eb8
0x4ec8	0x4ed8	0x4ee8	0x4ef8	0x4f0d	0x4f1d	0x4f2d	0x4f3d	0x4f4d	0x4f5d	0x4f6d	0x4f7d	0x4f8d	0x4f9d	0x4fad	0x4fbd	0x4fcd	0x4fdd	0x4fed	0x4ffd	0x5000
0x5010	0x5020	0x5030	0x5040	0x5050	0x5060	0x5070	0x5080	0x5090	0x50a0	0x50b0	0x50c0	0x50d0	0x50e0	0x50f0	0x5105	0x5115	0x5125	0x5135	0x5145	0x5155
0x5165	0x5175	0x5185	0x5195	0x51a5	0x51b5	0x51c5	0x51d5	0x51e5	0x51f5	0x520a	0x521a	0x522a	0x523a	0x524a	0x525a	0x526a	0x527a	0x528a	0x529a	0x52aa
0x52ba	0x52ca	0x52da	0x52ea	0x52fa	0x530f	0x531f	0x532f	0x533f	0x534f	0x535f	0x536f	0x537f	0x538f	0x539f	0x53af	0x53bf	0x53cf	0x53df	0x53ef	0x53ff
0x5401	0x5411	0x5421	0x5431	0x5441	0x5451	0x5461	0x5471	0x5481	0x5491	0x54a1	0x54b1	0x54c1	0x54d1	0x54e1	0x54f1	0x5504	0x5514	0x5524	0x5534	0x5544
0x5554	0x5564	0x5574	0x5584	0x5594	0x55a4	0x55b4	0x55c4	0x55d4	0x55e4	0x55f4	0x560b	0x561b	0x562b	0x563b	0x564b	0x565b	0x566b	0x567b	0x568b	0x569b
0x56ab	0x56bb	0x56cb	0x56db	0x56eb	0x56fb	0x570e	0x571e	0x572e	0x573e	0x574e	0x575e	0x576e	0x577e	0x578e	0x579e	0x57ae	0x57be	0x57ce	0x57de	0x57ee
0x57fe	0x5802	0x5812	0x5822	0x5832	0x5842	0x5852	0x5862	0x5872	0x5882	0x5892	0x58a2	0x58b2	0x58c2	0x58d2	0x58e2	0x58f2	0x5907	0x5917	0x5927	0x5937
0x5947	0x5957	0x5967	0x5977	0x5987	0x5997	0x59a7	0x59b7	0x59c7	0x59d7	0x59e7	0x59f7	0x5a08	0x5a18	0x5a28	0x5a38	0x5a48	0x5a58	0x5a68	0x5a78	0x5a88
0x5a98	0x5aa8	0x5ab8	0x5ac8	0x5ad8	0x5ae8	0x5af8	0x5b0d	0x5b1d	0x5b2d	0x5b3d	0x5b4d	0x5b5d	0x5b6d	0x5b7d	0x5b8d	0x5b9d	0x5bad	0x5bbd	0x5bcd	0x5bdd
0x5bed	0x5bfd	0x5c03	0x5c13	0x5c23	0x5c33	0x5c43	0x5c53	0x5c63	0x5c73	0x5c83	0x5c93	0x5ca3	0x5cb3	0x5cc3	0x5cd3	0x5ce3	0x5cf3	0x5d06	0x5d16	0x5d26
0x5d36	0x5d46	0x5d56	0x5d66	0x5d76	0x5d86	0x5d96	0x5da6	0x5db6	0x5dc6	0x5dd6	0x5de6	0x5df6	0x5e09	0x5e19	0x5e29	0x5e39	0x5e49	0x5e59	0x5e69	0x5e79
0x5e89	0x5e99	0x5ea9	0x5eb9	0x5ec9	0x5ed9	0x5ee9	0x5ef9	0x5f0c	0x5f1c	0x5f2c	0x5f3c	0x5f4c	0x5f5c	0x5f6c	0x5f7c	0x5f8c	0x5f9c	0x5fac	0x5fbc	0x5fcc
0x5fdc	0x5fec	0x5ffc	0x6003	0x6013	0x6023	0x6033	0x6043	0x6053	0x6063	0x6073	0x6083	0x6093	0x60a3	0x60b3	0x60c3	0x60d3	0x60e3	0x60f3	0x6106	0x6116
0x6126	0x6136	0x6146	0x6156	0x6166	0x6176	0x6186	0x6196	0x61a6	0x61b6	0x61c6	0x61d6	0x61e6	0x61f6	0x6209	0x6219	0x6229	0x6239	0x6249	0x6259	0x6269
0x6279	0x6289	0x6299	0x62a9	0x62b9	0x62c9	0x62d9	0x62e9	0x62f9	0x630c	0x631c	0x632c	0x633c	0x634c	0x635c	0x636c	0x637c	0x638c	0x639c	0x63ac	0x63bc
0x63cc	0x63dc	0x63ec	0x63fc	0x6402	0x6412	0x6422	0x6432	0x6442	0x6452	0x6462	0x6472	0x6482	0x6492	0x64a2	0x64b2	0x64c2	0x64d2	0x64e2	0x64f2	0x6507
0x6517	0x6527	0x6537	0x6547	0x6557	0x6567	0x6577	0x6587	0x6597	0x65a7	0x65b7	0x65c7	0x65d7	0x65e7	0x65f7	0x6608	0x6618	0x6628	0x6638	0x6648	0x6658
0x6668	0x6678	0x6688	0x6698	0x66a8	0x66b8	0x66c8	0x66d8	0x66e8	0x66f8	0x670d	0x671d	0x672d	0x673d	0x674d	0x675d	0x676d	0x677d	0x678d	0x679d	0x67ad
0x67bd	0x67cd	0x67dd	0x67ed	0x67fd	0x6801	0x6811	0x6821	0x6831	0x6841	0x6851	0x6861	0x6871	0x6881	0x6891	0x68a1	0x68b1	0x68c1	0x68d1	0x68e1	0x68f1
0x6904	0x6914	0x6924	0x6934	0x6944	0x6954	0x6964	0x6974	0x6984	0x6994	0x69a4	0x69b4	0x69c4	0x69d4	0x69e4	0x69f4	0x6a0b	0x6a1b	0x6a2b	0x6a3b	0x6a4b
0x6a5b	0x6a6b	0x6a7b	0x6a8b	0x6a9b	0x6aab	0x6abb	0x6acb	0x6adb	0x6aeb	0x6afb	0x6b0e	0x6b1e	0x6b2e	0x6b3e	0x6b4e	0x6b5e	0x6b6e	0x6b7e	0x6b8e	0x6b9e
0x6bae	0x6bbe	0x6bce	0x6bde	0x6bee	0x6bfe	0x6c00	0x6c10	0x6c20	0x6c30	0x6c40	0x6c50	0x6c60	0x6c70	0x6c80	0x6c90	0x6ca0	0x6cb0	0x6cc0	0x6cd0	0x6ce0
0x6cf0	0x6d05	0x6d15	0x6d25	0x6d35	0x6d45	0x6d55	0x6d65	0x6d75	0x6d85	0x6d95	0x6da5	0x6db5	0x6dc5	0x6dd5	0x6de5	0x6df5	0x6e0a	0x6e1a	0x6e2a	0x6e3a
0x6e4a	0x6e5a	0x6e6a	0x6e7a	0x6e8a	0x6e9a	0x6eaa	0x6eba	0x6eca	0x6eda	0x6eea	0x6efa	0x6f0f	0x6f1f	0x6f2f	0x6f3f	0x6f4f	0x6f5f	0x6f6f	0x6f7f	0x6f8f
0x6f9f	0x6faf	0x6fbf	0x6fcf	0x6fdf	0x6fef	0x6fff	0x7002	0x7012	0x7022	0x7032	0x7042	0x7052	0x7062	0x7072	0x7082	0x7092	0x70a2	0x70b2	0x70c2	0x70d2
0x70e2	0x70f2	0x7107	0x7117	0x7127	0x7137	0x7147	0x7157	0x7167	0x7177	0x7187	0x7197	0x71a7	0x71b7	0x71c7	0x71d7	0x71e7	0x71f7	0x7208	0x7218	0x7228
0x7238	0x7248	0x7258	0x7268	0x7278	0x7288	0x7298	0x72a8	0x72b8	0x72c8	0x72d8	0x72e8	0x72f8	0x730d	0x731d	0x732d	0x733d	0x734d	0x735d	0x736d	0x737d

```
0x738d 0x739d 0x73ad 0x73bd 0x73cd 0x73dd 0x73ed 0x73fd 0x7403 0x7413 0x7423 0x7433 0x7443 0x7453 0x7463 0x7473 0x7483 0x7493 0x74a3 0x74b3 0x74c3
0x74d3 0x74e3 0x74f3 0x7506 0x7516 0x7526 0x7536 0x7546 0x7556 0x7566 0x7576 0x7586 0x7596 0x75a6 0x75b6 0x75c6 0x75d6 0x75e6 0x75f6 0x7609 0x7619
0x7629 0x7639 0x7649 0x7659 0x7669 0x7679 0x7689 0x7699 0x76a9 0x76b9 0x76c9 0x76d9 0x76e9 0x76f9 0x770c 0x771c 0x772c 0x773c 0x774c 0x775c 0x776c
0x777c 0x778c 0x779c 0x77ac 0x77bc 0x77cc 0x77dc 0x77ec 0x77fc 0x7800 0x7810 0x7820 0x7830 0x7840 0x7850 0x7860 0x7870 0x7880 0x7890 0x78a0 0x78b0
0x78c0 0x78d0 0x78e0 0x78f0 0x7905 0x7915 0x7925 0x7935 0x7945 0x7955 0x7965 0x7975 0x7985 0x7995 0x79a5 0x79b5 0x79c5 0x79d5 0x79e5 0x79f5 0x7a0a
0x7a1a 0x7a2a 0x7a3a 0x7a4a 0x7a5a 0x7a6a 0x7a7a 0x7a8a 0x7a9a 0x7aaa 0x7aba 0x7aca 0x7ada 0x7aea 0x7afa 0x7b0f 0x7b1f 0x7b2f 0x7b3f 0x7b4f 0x7b5f
0x7b6f 0x7b7f 0x7b8f 0x7b9f 0x7baf 0x7bbf 0x7bcf 0x7bdf 0x7bef 0x7bff 0x7c01 0x7c11 0x7c21 0x7c31 0x7c41 0x7c51 0x7c61 0x7c71 0x7c81 0x7c91 0x7ca1
0x7cb1 0x7cc1 0x7cd1 0x7ce1 0x7cf1 0x7d04 0x7d14 0x7d24 0x7d34 0x7d44 0x7d54 0x7d64 0x7d74 0x7d84 0x7d94 0x7da4 0x7db4 0x7dc4 0x7dd4 0x7de4 0x7df4
0x7e0b 0x7e1b 0x7e2b 0x7e3b 0x7e4b 0x7e5b 0x7e6b 0x7e7b 0x7e8b 0x7e9b 0x7eab 0x7ebb 0x7ecb 0x7edb 0x7eeb 0x7efb 0x7f0e 0x7f1e 0x7f2e 0x7f3e 0x7f4e
0x7f5e 0x7f6e 0x7f7e 0x7f8e 0x7f9e 0x7fae 0x7fbe 0x7fce 0x7fde 0x7fee 0x7ffe 0x8002 0x8012 0x8022 0x8032 0x8042 0x8052 0x8062 0x8072 0x8082 0x8092
0x80a2 0x80b2 0x80c2 0x80d2 0x80e2 0x80f2 0x8107 0x8117 0x8127 0x8137 0x8147 0x8157 0x8167 0x8177 0x8187 0x8197 0x81a7 0x81b7 0x81c7 0x81d7 0x81e7
0x81f7 0x8208 0x8218 0x8228 0x8238 0x8248 0x8258 0x8268 0x8278 0x8288 0x8298 0x82a8 0x82b8 0x82c8 0x82d8 0x82e8 0x82f8 0x830d 0x831d 0x832d 0x833d
0x834d 0x835d 0x836d 0x837d 0x838d 0x839d 0x83ad 0x83bd 0x83cd 0x83dd 0x83ed 0x83fd 0x8403 0x8413 0x8423 0x8433 0x8443 0x8453 0x8463 0x8473 0x8483
0x8493 0x84a3 0x84b3 0x84c3 0x84e3 0x84f3 0x8506 0x8516 0x8526 0x8536 0x8546 0x8556 0x8566 0x8576 0x8586 0x8596 0x85a6 0x85b6 0x85c6 0x85d6
0x85e6 0x85f6 0x8609 0x8619 0x8629 0x8639 0x8649 0x8659 0x8669 0x8679 0x8689 0x8699 0x86a9 0x86b9 0x86c9 0x86d9 0x86e9 0x86f9 0x870c 0x871c 0x872c
0x873c 0x874c 0x875c 0x876c 0x877c 0x878c 0x879c 0x87ac 0x87bc 0x87cc 0x87dc 0x87ec 0x87fc 0x8800 0x8810 0x8820 0x8830 0x8840 0x8850 0x8860 0x8870
0x8880 0x8890 0x88a0 0x88b0 0x88c0 0x88d0 0x88e0 0x88f0 0x8905 0x8915 0x8925 0x8935 0x8945 0x8955 0x8965 0x8975 0x8985 0x8995 0x89a5 0x89b5 0x89c5
0x89d5 0x89e5 0x89f5 0x8a0a 0x8a1a 0x8a2a 0x8a3a 0x8a4a 0x8a5a 0x8a6a 0x8a7a 0x8a8a 0x8a9a 0x8aaa 0x8aba 0x8aca 0x8ada 0x8aea 0x8afa 0x8b0f 0x8b1f
0x8b2f 0x8b3f 0x8b4f 0x8b5f 0x8b6f 0x8b7f 0x8b8f 0x8b9f 0x8baf 0x8bbf 0x8bcf 0x8bdf 0x8bef 0x8bff 0x8c01 0x8c11 0x8c21 0x8c31 0x8c41 0x8c51 0x8c61
0x8c71 0x8c81 0x8c91 0x8ca1 0x8cb1 0x8cc1 0x8cd1 0x8ce1 0x8cf1 0x8d04 0x8d14 0x8d24 0x8d34 0x8d44 0x8d54 0x8d64 0x8d74 0x8d84 0x8d94 0x8da4 0x8db4
0x8dc4 0x8dd4 0x8de4 0x8df4 0x8e0b 0x8e1b 0x8e2b 0x8e3b 0x8e4b 0x8e5b 0x8e6b 0x8e7b 0x8e8b 0x8e9b 0x8eab 0x8ebb 0x8ecb 0x8edb 0x8eeb 0x8efb 0x8f0e
0x8f1e 0x8f2e 0x8f3e 0x8f4e 0x8f5e 0x8f6e 0x8f7e 0x8f8e 0x8f9e 0x8fae 0x8fbe 0x8fce 0x8fde 0x8fee 0x8ffe 0x9003 0x9013 0x9023 0x9033 0x9043 0x9053
0x9063 0x9073 0x9083 0x9093 0x90a3 0x90b3 0x90c3 0x90d3 0x90e3 0x90f3 0x9106 0x9116 0x9126 0x9136 0x9146 0x9156 0x9166 0x9176 0x9186 0x9196 0x91a6
0x91b6 0x91c6 0x91d6 0x91e6 0x91f6 0x9209 0x9219 0x9229 0x9239 0x9249 0x9259 0x9269 0x9279 0x9289 0x9299 0x92a9 0x92b9 0x92c9 0x92d9 0x92e9 0x92f9
0x930c 0x931c 0x932c 0x933c 0x934c 0x935c 0x936c 0x937c 0x938c 0x939c 0x93ac 0x93bc 0x93cc 0x93dc 0x93ec 0x93fc 0x9402 0x9412 0x9422 0x9432 0x9442
0x9452 0x9462 0x9472 0x9482 0x9492 0x94a2 0x94b2 0x94c2 0x94d2 0x94e2 0x94f2 0x9507 0x9517 0x9527 0x9537 0x9547 0x9557 0x9567 0x9577 0x9587 0x9597
0x95a7 0x95b7 0x95c7 0x95d7 0x95e7 0x95f7 0x9608 0x9618 0x9628 0x9638 0x9648 0x9658 0x9668 0x9678 0x9688 0x9698 0x96a8 0x96b8 0x96c8 0x96d8 0x96e8
0x96f8 0x970d 0x971d 0x972d 0x973d 0x974d 0x975d 0x976d 0x977d 0x978d 0x979d 0x97ad 0x97bd 0x97cd 0x97dd 0x97ed 0x97fd 0x9801 0x9811 0x9821 0x9831
0x9841 0x9851 0x9861 0x9871 0x9881 0x9891 0x98a1 0x98b1 0x98c1 0x98d1 0x98e1 0x98f1 0x9904 0x9914 0x9924 0x9934 0x9944 0x9954 0x9964 0x9974 0x9984
0x9994 0x99a4 0x99b4 0x99c4 0x99d4 0x99e4 0x99f4 0x9a0b 0x9a1b 0x9a2b 0x9a3b 0x9a4b 0x9a5b 0x9a6b 0x9a7b 0x9a8b 0x9a9b 0x9aab 0x9abb 0x9acb 0x9adb
0x9aeb 0x9afb 0x9b0e 0x9b1e 0x9b2e 0x9b3e 0x9b4e 0x9b5e 0x9b6e 0x9b7e 0x9b8e 0x9b9e 0x9bae 0x9bbe 0x9bce 0x9bde 0x9bee 0x9bfe 0x9c00 0x9c10 0x9c20
0x9c30 0x9c40 0x9c50 0x9c60 0x9c70 0x9c80 0x9c90 0x9ca0 0x9cb0 0x9cc0 0x9cd0 0x9ce0 0x9cf0 0x9d05 0x9d15 0x9d25 0x9d35 0x9d45 0x9d55 0x9d65 0x9d75
0x9d85 0x9d95 0x9da5 0x9db5 0x9dc5 0x9de5 0x9df5 0x9e0e 0x9e1a 0x9e2a 0x9e3a 0x9e4a 0x9e5a 0x9e6a 0x9e7a 0x9e8a 0x9e9a 0x9eaa 0x9eba 0x9eca
0x9eda 0x9eea 0x9efa 0x9f0f 0x9f1f 0x9f2f 0x9f3f 0x9f4f 0x9f5f 0x9f6f 0x9f7f 0x9f8f 0x9f9f 0x9faf 0x9fbf 0x9fcf 0x9fdf 0x9fef 0x9fff 0xa000 0xa010
0xa020 0xa030 0xa040 0xa050 0xa060 0xa070 0xa080 0xa090 0xa0a0 0xa0b0 0xa0c0 0xa0d0 0xa0e0 0xa0f0 0xa105 0xa115 0xa125 0xa135 0xa145 0xa155 0xa165
0xa175 0xa185 0xa195 0xa1a5 0xa1b5 0xa1c5 0xa1d5 0xa1e5 0xa1f5 0xa20a 0xa21a 0xa22a 0xa23a 0xa24a 0xa25a 0xa26a 0xa27a 0xa28a 0xa29a 0xa2aa 0xa2ba
0xa2ca 0xa2da 0xa2ea 0xa2fa 0xa30f 0xa31f 0xa32f 0xa33f 0xa34f 0xa35f 0xa36f 0xa37f 0xa38f 0xa39f 0xa3af 0xa3bf 0xa3cf 0xa3df 0xa3ef 0xa3ff 0xa401
0xa411 0xa421 0xa431 0xa441 0xa451 0xa461 0xa471 0xa481 0xa491 0xa4a1 0xa4b1 0xa4c1 0xa4d1 0xa4e1 0xa4f1 0xa504 0xa514 0xa524 0xa534 0xa544 0xa554
0xa564 0xa574 0xa584 0xa594 0xa5a4 0xa5b4 0xa5c4 0xa5d4 0xa5e4 0xa5f4 0xa60b 0xa61b 0xa62b 0xa63b 0xa64b 0xa65b 0xa66b 0xa67b 0xa68b 0xa69b 0xa6ab
0xa6bb 0xa6cb 0xa6db 0xa6eb 0xa6fb 0xa70e 0xa71e 0xa72e 0xa73e 0xa74e 0xa75e 0xa76e 0xa77e 0xa78e 0xa79e 0xa7ae 0xa7be 0xa7ce 0xa7de 0xa7ee 0xa7fe
0xa802 0xa812 0xa822 0xa832 0xa842 0xa852 0xa862 0xa872 0xa882 0xa892 0xa8a2 0xa8b2 0xa8c2 0xa8d2 0xa8e2 0xa8f2 0xa907 0xa917 0xa927 0xa937 0xa947
0xa957 0xa967 0xa977 0xa987 0xa997 0xa9a7 0xa9b7 0xa9c7 0xa9d7 0xa9e7 0xa9f7 0xaa08 0xaa18 0xaa28 0xaa38 0xaa48 0xaa58 0xaa68 0xaa78 0xaa88 0xaa98
0xaaa8 0xaab8 0xaac8 0xaad8 0xaae8 0xaaf8 0xab0d 0xab1d 0xab2d 0xab3d 0xab4d 0xab5d 0xab6d 0xab7d 0xab8d 0xab9d 0xabad 0xabbd 0xabcd 0xabdd 0xabed
0xabfd 0xac03 0xac13 0xac23 0xac33 0xac43 0xac53 0xac63 0xac73 0xac83 0xac93 0xaca3 0xacb3 0xacc3 0xacd3 0xace3 0xacf3 0xad06 0xad16 0xad26 0xad36
0xad46 0xad56 0xad66 0xad76 0xad86 0xad96 0xada6 0xadb6 0xadc6 0xadd6 0xade6 0xadf6 0xae09 0xae19 0xae29 0xae39 0xae49 0xae59 0xae69 0xae79 0xae89
0xae99 0xaea9 0xaeb9 0xaec9 0xaed9 0xaee9 0xaef9 0xaf0c 0xaf1c 0xaf2c 0xaf3c 0xaf4c 0xaf5c 0xaf6c 0xaf7c 0xaf8c 0xaf9c 0xafac 0xafbc 0xafcc 0xafdc
0xafec 0xaffc 0xb001 0xb011 0xb021 0xb031 0xb041 0xb051 0xb061 0xb071 0xb081 0xb0a1 0xb0b1 0xb0c1 0xb0d1 0xb0e1 0xb0f1 0xb104 0xb114 0xb124
0xb134 0xb144 0xb154 0xb164 0xb174 0xb184 0xb194 0xb1a4 0xb1b4 0xb1c4 0xb1d4 0xb1e4 0xb1f4 0xb20b 0xb21b 0xb22b 0xb23b 0xb24b 0xb25b 0xb26b 0xb27b
0xb28b 0xb29b 0xb2ab 0xb2bb 0xb2cb 0xb2db 0xb2eb 0xb2fb 0xb30e 0xb31e 0xb32e 0xb33e 0xb34e 0xb35e 0xb36e 0xb37e 0xb38e 0xb39e 0xb3ae 0xb3be 0xb3ce
0xb3de 0xb3ee 0xb3fe 0xb400 0xb410 0xb420 0xb430 0xb440 0xb450 0xb460 0xb470 0xb480 0xb490 0xb4a0 0xb4b0 0xb4c0 0xb4d0 0xb4e0 0xb4f0 0xb505 0xb515
0xb525 0xb535 0xb545 0xb555 0xb565 0xb575 0xb585 0xb595 0xb5a5 0xb5b5 0xb5c5 0xb5d5 0xb5e5 0xb5f5 0xb60a 0xb61a 0xb62a 0xb63a 0xb64a 0xb65a 0xb66a
0xb67a 0xb68a 0xb69a 0xb6aa 0xb6ba 0xb6ca 0xb6da 0xb6ea 0xb6fa 0xb70f 0xb71f 0xb72f 0xb73f 0xb74f 0xb75f 0xb76f 0xb77f 0xb78f 0xb79f 0xb7af 0xb7bf
0xb7cf 0xb7df 0xb7ef 0xb7ff 0xb803 0xb813 0xb823 0xb833 0xb843 0xb853 0xb863 0xb873 0xb883 0xb893 0xb8a3 0xb8b3 0xb8c3 0xb8d3 0xb8e3 0xb8f3 0xb906
0xb916 0xb926 0xb936 0xb946 0xb956 0xb966 0xb976 0xb986 0xb996 0xb9a6 0xb9b6 0xb9c6 0xb9d6 0xb9e6 0xb9f6 0xba09 0xba19 0xba29 0xba39 0xba49 0xba59
0xba69 0xba79 0xba89 0xba99 0xbaa9 0xbab9 0xbac9 0xbad9 0xbae9 0xbaf9 0xbb0c 0xbb1c 0xbb2c 0xbb3c 0xbb4c 0xbb5c 0xbb6c 0xbb7c 0xbb8c 0xbb9c 0xbbac
0xbbbc 0xbbcc 0xbbdc 0xbbec 0xbbfc 0xbc0c 0xbc12 0xbc22 0xbc32 0xbc42 0xbc52 0xbc62 0xbc72 0xbc82 0xbc92 0xbca2 0xbcb2 0xbcc2 0xbcd2 0xbce2 0xbcf2
0xbd07 0xbd17 0xbd27 0xbd37 0xbd47 0xbd57 0xbd67 0xbd77 0xbd87 0xbd97 0xbda7 0xbdb7 0xbdc7 0xbdd7 0xbde7 0xbdf7 0xbe08 0xbe18 0xbe28 0xbe38 0xbe48
0xbe58 0xbe68 0xbe78 0xbe88 0xbea8 0xbeb8 0xbec8 0xbed8 0xbee8 0xbef8 0xbf0d 0xbf1d 0xbf2d 0xbf3d 0xbf4d 0xbf5d 0xbf6d 0xbf7d 0xbf8d 0xbf9d
0xbfad 0xbfbd 0xbfcd 0xbfdd 0xbfed 0xbffd 0xc003 0xc013 0xc023 0xc033 0xc043 0xc053 0xc063 0xc073 0xc083 0xc093 0xc0a3 0xc0b3 0xc0c3 0xc0d3 0xc0e3
0xc0f3 0xc106 0xc116 0xc126 0xc136 0xc146 0xc156 0xc166 0xc176 0xc186 0xc196 0xc1a6 0xc1b6 0xc1c6 0xc1d6 0xc1e6 0xc1f6 0xc209 0xc219 0xc229 0xc239
0xc249 0xc259 0xc269 0xc279 0xc289 0xc299 0xc2a9 0xc2b9 0xc2c9 0xc2d9 0xc2e9 0xc2f9 0xc30c 0xc31c 0xc32c 0xc33c 0xc34c 0xc35c 0xc36c 0xc37c 0xc38c
0xc39c 0xc3ac 0xc3bc 0xc3cc 0xc3dc 0xc3ec 0xc3fc 0xc402 0xc412 0xc422 0xc432 0xc442 0xc452 0xc462 0xc472 0xc482 0xc492 0xc4a2 0xc4b2 0xc4c2 0xc4d2
0xc4e2 0xc4f2 0xc507 0xc517 0xc527 0xc537 0xc547 0xc557 0xc567 0xc577 0xc587 0xc597 0xc5a7 0xc5b7 0xc5c7 0xc5d7 0xc5e7 0xc5f7 0xc608 0xc618 0xc628
0xc638 0xc648 0xc658 0xc668 0xc678 0xc688 0xc698 0xc6a8 0xc6b8 0xc6c8 0xc6d8 0xc6e8 0xc6f8 0xc70d 0xc71d 0xc72d 0xc73d 0xc74d 0xc75d 0xc76d 0xc77d
0xc78d 0xc79d 0xc7ad 0xc7bd 0xc7cd 0xc7dd 0xc7ed 0xc7fd 0xc811 0xc821 0xc831 0xc841 0xc851 0xc861 0xc871 0xc881 0xc891 0xc8a1 0xc8b1 0xc8c1
0xc8d1 0xc8e1 0xc8f1 0xc904 0xc914 0xc924 0xc934 0xc944 0xc954 0xc964 0xc974 0xc984 0xc994 0xc9a4 0xc9b4 0xc9c4 0xc9d4 0xc9e4 0xc9f4 0xca0b 0xca1b
0xca2b 0xca3b 0xca4b 0xca5b 0xca6b 0xca7b 0xca8b 0xca9b 0xcaab 0xcabb 0xcacb 0xcadb 0xcaeb 0xcafb 0xcb0e 0xcb1e 0xcb2e 0xcb3e 0xcb4e 0xcb5e 0xcb6e
0xcb7e 0xcb8e 0xcb9e 0xcbae 0xcbbe 0xcbce 0xcbde 0xcbee 0xcbfe 0xcc00 0xcc10 0xcc20 0xcc30 0xcc40 0xcc50 0xcc60 0xcc70 0xcc80 0xcc90 0xcca0 0xccb0
0xccc0 0xccd0 0xcce0 0xccf0 0xcd05 0xcd15 0xcd25 0xcd35 0xcd45 0xcd55 0xcd65 0xcd75 0xcd85 0xcd95 0xcda5 0xcdb5 0xcdc5 0xcdd5 0xcde5 0xcdf5 0xce0a
0xce1a 0xce2a 0xce3a 0xce4a 0xce5a 0xce6a 0xce7a 0xce8a 0xce9a 0xceaa 0xceba 0xceca 0xceda 0xceea 0xcefa 0xcf0f 0xcf1f 0xcf2f 0xcf3f 0xcf4f 0xcf5f
0xcf6f 0xcf7f 0xcf8f 0xcf9f 0xcfaf 0xcfbf 0xcfcf 0xcfdf 0xcfef 0xcfff 0xd002 0xd012 0xd022 0xd032 0xd042 0xd052 0xd062 0xd072 0xd082 0xd092 0xd0a2
0xd0b2 0xd0c2 0xd0d2 0xd0e2 0xd0f2 0xd107 0xd117 0xd127 0xd137 0xd147 0xd157 0xd167 0xd177 0xd187 0xd197 0xd1a7 0xd1b7 0xd1c7 0xd1d7 0xd1e7 0xd1f7
0xd208 0xd218 0xd228 0xd238 0xd248 0xd258 0xd268 0xd278 0xd288 0xd298 0xd2a8 0xd2b8 0xd2c8 0xd2d8 0xd2e8 0xd2f8 0xd30d 0xd31d 0xd32d 0xd33d 0xd34d
0xd35d 0xd36d 0xd37d 0xd38d 0xd39d 0xd3ad 0xd3bd 0xd3cd 0xd3dd 0xd3ed 0xd3fd 0xd403 0xd413 0xd423 0xd433 0xd443 0xd453 0xd463 0xd473 0xd483 0xd493
0xd4a3 0xd4b3 0xd4c3 0xd4d3 0xd4e3 0xd4f3 0xd506 0xd516 0xd526 0xd536 0xd546 0xd556 0xd566 0xd576 0xd586 0xd596 0xd5a6 0xd5b6 0xd5c6 0xd5d6 0xd5e6
0xd5f6 0xd609 0xd619 0xd629 0xd639 0xd649 0xd659 0xd669 0xd679 0xd689 0xd699 0xd6a9 0xd6b9 0xd6c9 0xd6d9 0xd6e9 0xd6f9 0xd70c 0xd71c 0xd72c 0xd73c
0xd74c 0xd75c 0xd76c 0xd77c 0xd78c 0xd79c 0xd7ac 0xd7bc 0xd7cc 0xd7dc 0xd7ec 0xd7fc 0xd800 0xd810 0xd820 0xd830 0xd840 0xd850 0xd860 0xd870 0xd880
0xd890 0xd8a0 0xd8b0 0xd8c0 0xd8d0 0xd8e0 0xd8f0 0xd905 0xd915 0xd925 0xd935 0xd945 0xd955 0xd965 0xd975 0xd985 0xd995 0xd9a5 0xd9b5 0xd9c5 0xd9d5
0xd9e5 0xd9f5 0xda0a 0xda1a 0xda2a 0xda3a 0xda4a 0xda5a 0xda6a 0xda7a 0xda8a 0xda9a 0xdaaa 0xdaba 0xdaca 0xdada 0xdaea 0xdafa 0xdb0f 0xdb1f 0xdb2f
0xdb3f 0xdb4f 0xdb5f 0xdb6f 0xdb7f 0xdb8f 0xdb9f 0xdbaf 0xdbbf 0xdbcf 0xdbdf 0xdbef 0xdbff 0xdc01 0xdc11 0xdc21 0xdc31 0xdc41 0xdc51 0xdc61 0xdc71
0xdc81 0xdc91 0xdca1 0xdcb1 0xdcc1 0xdcd1 0xdce1 0xdcf1 0xdd04 0xdd14 0xdd24 0xdd34 0xdd44 0xdd54 0xdd64 0xdd74 0xdd84 0xdd94 0xdda4 0xddb4 0xddc4
0xddd4 0xdde4 0xddf4 0xde0b 0xde1b 0xde2b 0xde3b 0xde4b 0xde5b 0xde6b 0xde7b 0xde8b 0xde9b 0xdeab 0xdebb 0xdecb 0xdedb 0xdeeb 0xdefb 0xdf0e 0xdf1e
0xdf2e 0xdf3e 0xdf4e 0xdf5e 0xdf6e 0xdf7e 0xdf8e 0xdf9e 0xdfae 0xdfbe 0xdfce 0xdfde 0xdfee 0xdffe 0xe001 0xe011 0xe021 0xe031 0xe041 0xe051 0xe061
0xe071 0xe081 0xe091 0xe0a1 0xe0b1 0xe0c1 0xe0d1 0xe0e1 0xe0f1 0xe104 0xe114 0xe124 0xe134 0xe144 0xe154 0xe164 0xe174 0xe184 0xe194 0xe1a4 0xe1b4
0xe1c4 0xe1d4 0xe1e4 0xe1f4 0xe20b 0xe21b 0xe22b 0xe23b 0xe24b 0xe25b 0xe26b 0xe27b 0xe28b 0xe29b 0xe2ab 0xe2bb 0xe2cb 0xe2db 0xe2eb 0xe2fb 0xe30e
0xe31e 0xe32e 0xe33e 0xe34e 0xe35e 0xe36e 0xe37e 0xe38e 0xe39e 0xe3ae 0xe3be 0xe3ce 0xe3de 0xe3ee 0xe3fe 0xe400 0xe410 0xe420 0xe430 0xe440 0xe450
0xe460 0xe470 0xe480 0xe490 0xe4a0 0xe4b0 0xe4c0 0xe4d0 0xe4e0 0xe4f0 0xe505 0xe515 0xe525 0xe535 0xe545 0xe555 0xe565 0xe575 0xe585 0xe595 0xe5a5
0xe5b5 0xe5c5 0xe5d5 0xe5e5 0xe5f5 0xe60a 0xe61a 0xe62a 0xe63a 0xe64a 0xe65a 0xe66a 0xe67a 0xe68a 0xe69a 0xe6aa 0xe6ba 0xe6ca 0xe6da 0xe6ea 0xe6fa
0xe70f 0xe71f 0xe72f 0xe73f 0xe74f 0xe75f 0xe76f 0xe77f 0xe78f 0xe79f 0xe7af 0xe7bf 0xe7cf 0xe7df 0xe7ef 0xe7ff 0xe803 0xe813 0xe823 0xe833 0xe843
0xe853 0xe863 0xe873 0xe883 0xe893 0xe8a3 0xe8b3 0xe8c3 0xe8d3 0xe8e3 0xe8f3 0xe906 0xe916 0xe926 0xe936 0xe946 0xe956 0xe966 0xe976 0xe986 0xe996
0xe9a6 0xe9b6 0xe9c6 0xe9d6 0xe9e6 0xe9f6 0xea09 0xea19 0xea29 0xea39 0xea49 0xea59 0xea69 0xea79 0xea89 0xea99 0xeaa9 0xeab9 0xeac9 0xead9 0xeae9
0xeaf9 0xeb0c 0xeb1c 0xeb2c 0xeb3c 0xeb4c 0xeb5c 0xeb6c 0xeb7c 0xeb8c 0xeb9c 0xebac 0xebbc 0xebcc 0xebdc 0xebec 0xebfc 0xec02 0xec12 0xec22 0xec32
0xec42 0xec52 0xec62 0xec72 0xec82 0xec92 0xeca2 0xecb2 0xecc2 0xecd2 0xece2 0xecf2 0xed07 0xed17 0xed27 0xed37 0xed47 0xed57 0xed67 0xed77 0xed87
0xed97 0xeda7 0xedb7 0xedc7 0xedd7 0xede7 0xedf7 0xee08 0xee18 0xee28 0xee38 0xee48 0xee58 0xee68 0xee78 0xee88 0xee98 0xeea8 0xeeb8 0xeec8 0xeed8
0xeee8 0xeef8 0xef0d 0xef1d 0xef2d 0xef3d 0xef4d 0xef5d 0xef6d 0xef7d 0xef8d 0xef9d 0xefad 0xefbd 0xefcd 0xefdd 0xefed 0xeffd 0xf000 0xf010 0xf020
0xf030 0xf040 0xf050 0xf060 0xf070 0xf080 0xf090 0xf0a0 0xf0b0 0xf0c0 0xf0d0 0xf0e0 0xf0f0 0xf105 0xf115 0xf125 0xf135 0xf145 0xf155 0xf165 0xf175
0xf185 0xf195 0xf1a5 0xf1b5 0xf1c5 0xf1d5 0xf1e5 0xf1f5 0xf20a 0xf21a 0xf22a 0xf23a 0xf24a 0xf25a 0xf26a 0xf27a 0xf28a 0xf29a 0xf2aa 0xf2ba 0xf2ca
0xf2da 0xf2ea 0xf2fa 0xf30f 0xf31f 0xf32f 0xf33f 0xf34f 0xf35f 0xf36f 0xf37f 0xf38f 0xf39f 0xf3af 0xf3bf 0xf3cf 0xf3df 0xf3ef 0xf3ff 0xf401 0xf411
0xf421 0xf431 0xf441 0xf451 0xf461 0xf471 0xf481 0xf491 0xf4a1 0xf4b1 0xf4c1 0xf4d1 0xf4e1 0xf4f1 0xf504 0xf514 0xf524 0xf534 0xf544 0xf554 0xf564
0xf574 0xf584 0xf594 0xf5a4 0xf5b4 0xf5c4 0xf5d4 0xf5e4 0xf5f4 0xf60b 0xf61b 0xf62b 0xf63b 0xf64b 0xf65b 0xf66b 0xf67b 0xf68b 0xf69b 0xf6ab 0xf6bb
0xf6cb 0xf6db 0xf6eb 0xf6fb 0xf70e 0xf71e 0xf72e 0xf73e 0xf74e 0xf75e 0xf76e 0xf77e 0xf78e 0xf79e 0xf7ae 0xf7be 0xf7ce 0xf7de 0xf7ee 0xf7fe 0xf802
0xf812 0xf822 0xf832 0xf842 0xf852 0xf862 0xf872 0xf882 0xf892 0xf8a2 0xf8b2 0xf8c2 0xf8d2 0xf8e2 0xf8f2 0xf907 0xf917 0xf927 0xf937 0xf947 0xf957
0xf967 0xf977 0xf987 0xf997 0xf9a7 0xf9b7 0xf9c7 0xf9d7 0xf9e7 0xfa08 0xfa18 0xfa28 0xfa38 0xfa48 0xfa58 0xfa68 0xfa78 0xfa88 0xfa98 0xfaa8
0xfab8 0xfac8 0xfad8 0xfae8 0xfaf8 0xfb0d 0xfb1d 0xfb2d 0xfb3d 0xfb4d 0xfb5d 0xfb6d 0xfb7d 0xfb8d 0xfb9d 0xfbad 0xfbbd 0xfbcd 0xfbdd 0xfbed 0xfbfd
0xfc03 0xfc13 0xfc23 0xfc33 0xfc43 0xfc53 0xfc63 0xfc73 0xfc83 0xfc93 0xfca3 0xfcb3 0xfcc3 0xfcd3 0xfce3 0xfcf3 0xfd06 0xfd16 0xfd26 0xfd36 0xfd46
0xfd56 0xfd66 0xfd76 0xfd86 0xfd96 0xfda6 0xfdb6 0xfdc6 0xfdd6 0xfde6 0xfdf6 0xfe09 0xfe19 0xfe29 0xfe39 0xfe49 0xfe59 0xfe69 0xfe79 0xfe89 0xfe99
0xfea9 0xfeb9 0xfec9 0xfed9 0xfee9 0xfef9 0xff0c 0xff1c 0xff2c 0xff3c 0xff4c 0xff5c 0xff6c 0xff7c 0xff8c 0xff9c 0xffac 0xffbc 0xffcc 0xffdc 0xffec
0xfffc
```

D Side-channel Injection Attack to Realize ΔR

As mentioned in the Proof of Theorem 1, condition (A) gives us the same differential sequence as the following condition does,

$$\Delta K = (K_1, ..., K_8) + (K'_1, ..., K'_8) = (0,0,0,0,0,0,0,0)$$
$$\Delta P = P_1 + P'_{i'} = 0$$
$$\Delta R = (R_1, ..., R_8) + (R'_1, ..., R'_8) = (0,0,0,H,0,0,0,0),$$

Therefore, to create the difference between R_4 and R'_4 (or between $f^{-1}(R_2 \boxminus u_1)$ and $f^{-1}(R'_2 \boxminus u'_1)$), one obvious way is to start with two instances initialized with the same IVs and keys and then mount side-channel injection attack, where the attacker simply injects H to the victim register, e.g., R_4 or $f^{-1}(R_2 \boxminus u_1)$, of one instance any time before the execution of the last round of encryption/decryption. Note that the *preparation through injection* gives the attacker no time/memory penalty, i.e., the overall time/memory complexity of the attack is dominated by that of the *key recovery phase*.