Differential Cryptanalysis of K-Cipher

1st Mohammad Mahzoun
Eindhoven University of Technology, the Netherlands
m.mahzoun@tue.nl

1st Liliya Kraleva
imec-COSIC, KU Leuven, Belgium
liliya.kraleva@esat.kuleuven.be

2nd Raluca Posteuca
imec-COSIC, KU Leuven, Belgium
raluca.posteuca@esat.kuleuven.be

3rd Tomer Ashur
imec-COSIC, KU Leuven, Belgium; and
Eindhoven University of Technology, the Netherlands
tomer.ashur@esat.kuleuven.be

***Abstract*—K-Cipher is an ultra low latency block cipher with variable-length parameters designed by Intel Labs. In this work, we analyze the security of K-Cipher and propose a differential cryptanalysis attack with the complexity of $2^{29.7}$ for a variant of K-Cipher with state size $n = 24$ bits state and block size $m = 8$ bits. Our attack recovers the secret key and secret randomizer values with a total length of 240 bits in $\sim$ 30 minutes on a standard desktop machine. We show that it is possible to extend the same attack for an arbitrary set of parameters.**

I. Introduction

K-Cipher [1] is a low latency parametrizable block cipher proposed by researchers from Intel Labs at IEEE ISCC 2020. The motivation for designing K-Cipher is that current low latency ciphers such as PRINCE [2], PRINCEv2 [3], MANTIS [4], QARMA [5] and SPEEDY [6], as well as the proposed ciphers at the NIST Lightweight competition, do not offer a wide range of values for the state size and block size or are not efficient enough. Despite the fact that it is published recently, K-Cipher attracted a lot of attention and is being used in other works because of its resistance to known ciphertext attacks, its small block size, and extremely low latency. An instance of such works is a new memory safety mechanism called C^3 [7] which is proposed in 2021 and uses K-Cipher as its underlying block cipher to encrypt the pointers to the memory blocks.

In this work, we analyze the security of K-Cipher against differential cryptanalysis - an attack that the authors claim the cipher is secure against. We propose a full key-recovery attack on the 24-bit input with the 8-bit SBox version and the 128-bit input and 16-bit SBox version, as well as an evaluation of some other possible parameters. Our attack is a chosen-plaintext attack and needs the corresponding encryption of 2^{14} pairs of chosen plaintexts. The complexity of our attack is $2^{29.7}$ encryptions/decryptions for all variants of K-Cipher with block size $m = 8$ and can fully recover all the secret values. We run our attack on a standard desktop machine and recover all 240 secret bits in $\sim$ 30 minutes.

The paper is structured as follows. In Section II, we introduce notations, differential cryptanalysis, and the description of K-Cipher. In Section III we describe the attack. We first explain in Section III-A a technique to construct characteristics independently of the random values added in the SBox layer. Further, in Section III-B we extend the characteristics to a key-recovery attack and recover the secret data. In Section III-C we analyze the complexity of our attack. Finally, we write our conclusion in Section IV.

This work was supported by CyberSecurity Research Flanders with reference number VR20192203. In addition it was supported by Intel through the Intel Project on Cryptographic Frontiers. Liliya Kraleva is a supported by a Ph.D. Fellowship from the Research Foundation — Flanders (FWO). Tomer Ashur is an FWO post-doctoral fellow under Grant Number 12ZH420N.

II. Preliminaries

A. Notation

We denote a binary string B with length n by $B = b_{n-1}b_{n-2}\dots b_0$ where each $b_i \in \{0,1\}$. Binary strings are used to describe the states of the K-Cipher. Each state B is the concatenation of several consecutive blocks $B[i]_{1\leq i\leq \frac{n}{m}}$ of size m. In other words, $B = B[1]\|B[2]\|\dots\|B[m] = b_{n-1}b_{n-2}\dots b_0$. If the state size n is not the multiple of the block size m, the size of the last block is equal to the $n \bmod m$. The SBoxes of K-Cipher are indexed from left to right as $S[1], S[2], \dots, S[m]$ and the input of $S[i]$ is the block $B[i]$. A binary sequence with zeros in all positions except for the i_{th} position is denoted by $e_i = [000\dots1\dots000]$.

In the rest of this paper, P is the input, C is the output, and $\mathcal{X}_i$'s are the intermediate states. The binary operations $\lll$ and $\ggg$ are used to show circular left shift and circular right shift. We define $+_m$ as block-wise modular addition over $GF(2^m)$. For two binary strings B_1 and B_2 with length km, $B_1 +_m B_2$ divides both B_1 and B_2 into k blocks of size m, then adds corresponding blocks modulo 2^m, and concatenates the resulting blocks in the same order. We define $-_m$ the same way as $+_m$ but for subtraction.

B. Differential cryptanalysis

Differential cryptanalysis is one of the most powerful and used cryptanalysis techniques against symmetric primitives. It is proposed by Biham and Shamir and is the first attack that fully breaks the DES algorithm [8]. This technique has been largely analyzed and further developed, leading to attacks like truncated differentials [9], the boomerang attack [10] or impossible differentials [11]. Differential cryptanalysis is a statistical chosen-plaintext attack that follows the difference $a = m \oplus m'$ between two plaintexts m and m', usually differing in only a few bits. Here we look only into XOR

difference, however other group operations can also be analyzed. The corresponding difference between the ciphertext can be exploited to reveal information about the secret key.

A differential characteristic over n rounds of an iterated cipher is defined as the sequence of intermediate differences $(a = a_1, a_2 \dots a_n = b)$, where each (a_i, a_{i+1}) represents the input and output difference of one round. In order to calculate the probability of a one-round transition (a_i, a_{i+1}), we look into the non-linear round operation, typically an SBox. A Difference Distribution Table (DDT) over a given SBox contains the information of how many times each input difference leads to each output difference and is calculated over all possible inputs. The more probable transformations from a DDT table are used to construct differential characteristics over more rounds. Then, assuming that the rounds are independent and the keys are uniformly distributed, the Expected Differential Probability (EDP) of a characteristic is computed by multiplying the probabilities of each individual round:

$$EDP(a_1, a_2, \dots a_n) = \prod_i^n DP(a_{i-1}, a_i) \tag{1}$$

where DP represents the probability differential characteristics for one round.

In practice, only the input and output differences can be observed without considering the intermediate differences. We define the set of all differential characteristics with input difference a and output difference b as a differential (a, b). The probability of a differential (a, b) is the sum of the probabilities of all differential characteristics having input difference a and output difference b.

a) Key-recovery: To recover the secret values used in round r, we use an $r-1$ round differential characteristic (distinguisher) with probability p. First, we select $N = \frac{4}{p}$ random pair plaintexts with the expected input difference and their corresponding ciphertexts. The constant 4 is chosen to guarantee the discarding of wrong candidate keys and ensure the success of the attack. The next step is to guess the secret values and verify the guess using the distinguisher. For each key guess, a counter is kept and incremented if the expected difference is satisfied after partial decryption using the candidate values. The set of keys with the largest counter is potential candidates for the secret values.

C. Rotational property

Word rotation is a commonly used operation for symmetric-key algorithms. The rotational property is linear with respect to cyclic rotation and XOR, but is not linear with respect to modular addition. While we have

$$\Pr\left[(x \lll l) \oplus (y \lll l) = (x \oplus y) \lll l\right] = 1 \tag{2}$$

and

$$\Pr\left[(x \lll l_1) \lll l_2 = (x \lll l_2) \lll l_1\right] = 1 \tag{3}$$

For modular addition we have the following probability [12]:

$$\begin{aligned} Pr[(x +_m y) \lll l = (x \lll l) +_m (y \lll l)] \\ = \frac{1}{4}(1 + 2^{l-n} + 2^{-l} + 2^{-n}) \end{aligned} \tag{4}$$

where l is the rotation constant and n is the size of the word.

D. Specifications of K-Cipher

K-Cipher is a low latency parameterizable block cipher proposed by Intel Research Labs [1], [13] in 2020. The original design of K-Cipher, called Flex flow, uses a deterministic SBox layer in its design. However, the authors suggested that Flex flow is insecure and proposed a new version, called CPA flow, with a nondeterministic SBox layer. We will omit the analysis of the Flex flow and focus on analyzing the CPA flow which is believed to be secure. We analyze the variant of K-Cipher with state size $n = 24$. A discussion on the different choices of parameters is presented in Section III-E after the key-recovery attack is explained.

a) Parameters selection.: K-Cipher has a state size of $24 \leq n \leq 1024$ and computes other parameters such as length of the keys, randomizers, and SBoxes using the state size n. The function `GetSBoxLengths(n)` is used to compute the length of the SBox m which we also refer to as block size. For our case of analysis with $n = 24$, the block size is $m = 8$. The length of the secret values is computed using the function `GetKeyLength(n, _)`. The length of the secret for a state of length n is $l + 6n$ where l is a constant computed using the function `GetFlexKeyLength(n)`. For $n = 24$, we have $l = 96$ and the key length is $96 + 6 \cdot 24 = 240$ bits.

b) Key and Randomizers.: The secret values used in K-Cipher are the master key $\mathcal{K}$, and the randomizers $\mathcal{R}$. The master key is used to generate round keys and the goal of the randomizers is to randomize the SBox layer so that each SBox behaves differently. The round keys are denoted by k_0, k_1, k_2, k_3 where k_0, k_1, k_2 are derived from a key schedule and k_3 is a known permutation of k_2. The randomizers are indexed by $r_0^j, r_1^j \quad 0 \leq j \leq 2$. For the state size $n \leq 32$, k_0, k_1, k_2 are substrings of $\mathcal{K}$ and are independent random values while for $n > 32$, the key schedule is the Flex flow. The idea of the key schedule algorithm for states of size $n > 32$ is described in Section III-E. For the more detailed description of the key schedule, we refer to [1].

c) Algorithm.: The encryption algorithm is depicted in Figure 1. The main component is called aggressive adder which is the modular addition of the round key over $GF(2^m)$ with the state followed by a permutation layer. The result of the aggressive adder function then goes to the SBox layer. The SBox layer of the j^{th} round XORs the m-bit block with the randomizer $r_0[j]$, then the multiplicative inverse of each resulted block in $GF(2^m)$ is added to randomizer $r_1[j]$ using modular addition. Finally, each block is cyclically rotated by 2. In the case where n is not a multiple of m, the last z bits are treated as a block of size z, and all operations are performed in $GF(2^z)$.

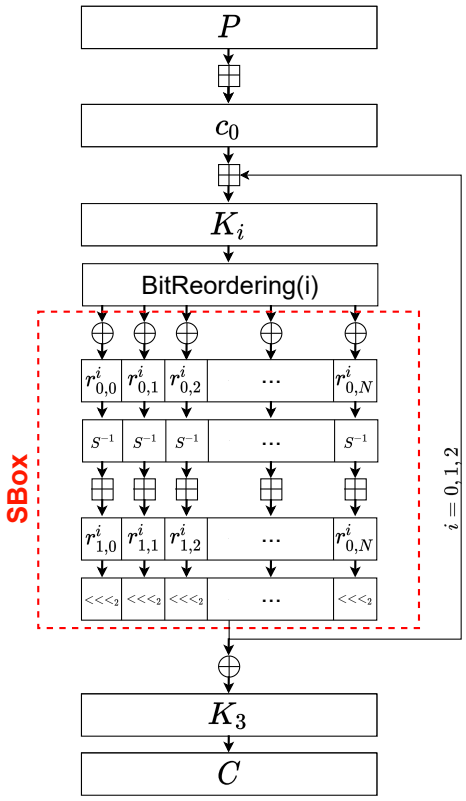

Fig. 1. The encryption algorithm of K-Cipher. First, the constant c is added to the input P using modular addition. Then 3 rounds are performed, each consisting of modular addition with the round key followed by a randomized SBox layer. Finally, another key value is XORed to the state and the ciphertext C is returned.

d) Security claims.: The authors claim that the security is independent of the size of the input and perform experiments over 24-bit state. The security over all other input sizes is then bounded by these results. They report the existence of differentials with a maximum probability of $2^{-17.61}$ over 3 rounds and $2^{-20.54}$ over 4 rounds and suggested that $r \geq 2$ is enough to resist differential cryptanalysis attack.

III. Differential Cryptanalysis of K-Cipher

In this section, we analyze the security of K-Cipher against differential cryptanalysis. First, we explain how to find differential characteristics of secret SBoxes used in K-Cipher. Next, we describe how to recover the secret key and the secret randomizers. Finally, we analyze the complexity and the running time of the attack.

A. Generating differential characteristics

The SBox transformation used in K-Cipher is the following function mapping $3m$ bits to m bits.

$$S(x, r_0^i[j], r_1^i[j]) = \left[(x \oplus r_0^i[j]) +_m r_1^i[j]\right] \lll 2 \qquad (5)$$

The use of so much randomness is motivated by improving the resistance of the SBox layer towards differential cryptanalysis by adding extra entropy. However, it is easy to see that for each $y \in GF(2^m)$, there are 2^{2m} tuples of (x, r_0, r_1) such that $S(x, r_0, r_1) = y$. Therefore, the construction of SBox leads to duplicate equivalent keys.

Our goal is to find differential characteristics such that the pair of inputs differ at exactly one bit, and their corresponding 2-round encryptions also differ at exactly one bit. Since xor with r_0 does not change the difference, we create DDT tables for $x^{-1} +_m r_1$. For each 8-bit possible value r_1, we create a 256 DDT table. Then all DDTs are combined into one General DDT, where the position (i, j) in GDDT will be non-zero if and only if it is non-zero for each of the 256 DDT tables. To ensure that our characteristic is valid for any choice of r_1^i we use the GDDT to create the needed characteristics. All 1-bit transitions of the GDDT are summarized in Table I, where the minimum probability is 2^{-7}. The characteristics used in attacking each round of the key-recovery attack are shown in Table II. The expected probability of a characteristic depends on the SBox and the modular addition and is calculated as the product of the one-round probabilities. The probability of 1-bit transition for modular addition over $GF(2^8)$ is $2^{-0.83}$. Therefore, A two-round characteristic has the probability $\Pr \leq 2^{-7} \cdot 2^{-0.83} \cdot 2^{-7} \cdot 2^{-0.83} = 2^{-15.66}$. All probabilities in the tables are verified empirically.

TABLE I
A part of the GDDT consisting only of the differences that satisfy our requirements for one active bit in the input and the output difference. The entry (i, j) gives the average number of times the input difference holds over all possible values of the randomizer for an input difference i and an output difference j. The probability is then the number divided by 256. For simplicity the unfilled values of the table are zeroes.

	1	2	4	8	16	32	64	128
1	2							
2			5					
4		3			3			
8	2			2				
32								2
64								2
128						3		

B. Description of the attack

We divide the attack into three phases for simplicity and in each phase, we recover additional bits of the unknown parameters. Each phase can also be seen as an individual attack over a similar primitive. The first phase targets the last round of the algorithm and obtains 2^3 indistinguishable candidates for k_3, k_2 and r_1^2. Phase two extends to the second round of the algorithm and recovers the randomizer values r_0^2 and r_1^1. Next phase repeats the same attack and recovers r_0^1 and $(r_1^0 \lll 2) +_m k_1$, where the exact values of r_1^0 and k_1 are not needed. Finally, the values of r_0^0 and k_0 are guessed.The details of each phase of the attack are described in Sections III-B1 to III-B3.

1) Phase 1:Recovering k_3, k_2 and r_1^2.: We construct 3 characteristics, each having a different active output block

after Bit-reordering(2). For each characteristics $i, i = 0, 1, 2$, we iterate over all possible 8-bit values of $k_3[i]$ and $r_1^2[i]$ and partially decrypt until the xor addition of r_0^2 of the SBox layer. The empirical difference is then verified with the difference of the corresponding characteristic in order to filter the correct values. If the partial decryption results in the expected difference, a counter associated with the current values of $k_3[i]$ and $r_1^2[i]$ is incremented. The counter with the highest value is the most likely assignment. The partial decryption is described in Algorithm 2 and the full recovery can be seen in Algorithm 1. The mathematical structure of K-Cipher implies that there are two pairs with the same highest counter value, and these are indistinguishable. The existence of such equivalent candidates is proven in Lemma III.1. This results in 2^3 candidate pairs of k_3 and r_0^2 and the choice of any of them will result in the same ciphertext. This is possible only because for $n \leq 32$ the round keys are directly obtained from the master key and there is no key schedule, therefore the keys and randomizers are independent. In the case of $n > 32$, keys are derived from a key schedule, which is bijective and hence invertible. By deriving 2^m candidate last round keys leads to 2^m different candidate master keys, the correct value of which can be guessed with probability 2^{-m}. After recovering k_3, we can recover the value of k_2 using the inverse of the BitReordering$(k_2, 3)$ permutation.

Lemma III.1. *The size of the set of candidates returned by the Algorithm 1 is always 2.*

Proof. The proof contains two parts. The first part is when $t = 3$ and the second part when $t \leq 2$. When $t = 3$, the set of candidates has the form $\{(k, r), (k +_m 2, r +_m 128)\}$ due the structure of the function. If $k \equiv 0 \mod 4$, then $k = k_7k_6k_5k_4k_300$ and $k +_m 2 = k_7k_6k_5k_4k_310$. The first step in the Algorithm 2 is xored with the input. Let us denote the input with $x = x_7x_6x_5x_4x_3x_2x_1x_0$. We have $y = x \oplus k = y_7y_6y_5y_4y_3y_2x_1x_0$ and $y' = x \oplus (k +_m 2) = y_7y_6y_5y_4y_3y_2(\neg x_1)x_0$. Let $r = r_7r_6r_5r_4r_3r_2r_1r_0$ and $r +_m 128 = (\neg r_7)r_6r_5r_4r_3r_2r_1r_0$. We have the following:

$$\begin{aligned}&(y \ggg 2) -_m r\\ &= x_1x_0y_7y_6y_5y_4y_3y_2 -_m r_7r_6r_5r_4r_3r_2r_1r_0\end{aligned} \tag{6}$$

$$\begin{aligned}&(y' \ggg 2) -_m (r +_m 128)\\ &= (\neg x_1)x_0y_7y_6y_5y_4y_3y_2 -_m (\neg r_7)r_6r_5r_4r_3r_2r_1r_0\end{aligned} \tag{7}$$

Since the most significant bit of the both operands is complemented, the result of the modular subtraction will be the same. In the case of $k \equiv 1 \mod 4$ the same argument applies because only the value of x_0 is complemented after the xor and does not affect the rest of the computations. When $t \leq 2$, the set of candidates has the form $\{(k, r), (k +_m 128, r +_m 128)\}$ because of the structure of the function. Let us define $k = k_7k_6k_5k_4k_3k_2k_1k_0$ and $k +_m 128 = (\neg k_7)k_6k_5k_4k_3k_2k_1k_0$. Like the previous part of the proof, we denote the input with $x = x_7x_6x_5x_4x_3x_2x_1x_0$. Let $y = x \oplus k = y_7y_6y_5y_4y_3y_2y_1y_0$ and $y' = x \oplus (k +_m 128) = (\neg y_7)y_6y_5y_4y_3y_2y_1y_0$. Let

Algorithm 1: Algorithm of the differential cryptanalysis attack, the return value for phase 1 is $k_3[j], r_1^2[j]$, for phase 2 is $r_0^2[j], (r_1^1 \ggg 2) +_m k_2$, and for phase 3 is $r_0^1[j], (r_1^0 \ggg 2) +_m k_1$

Input: Characteristic $\mathcal{C}$ and the target round $1 \leq \mathcal{R} \leq 3$.
Output: Set of all candidates for the secret of round $\mathcal{R}$.

$\mathcal{S} := \{[(P_{i,1}, C_{i,1}), (P_{i,2}, C_{i,2})] \mid i \leq 2^{16}$ and $P_{i,1} \oplus P_{i,2} = e_{\text{input difference}}\}$.
$t := \mathcal{R} - 1$
for $0 \leq i \leq |\mathcal{S}|$ **do**
  for *each* 8*-bit strings* k *and* r **do**
    if $\mathcal{R} \leq 2$ **then**
      $C_{i,1} :=$ Decryption of $C_{i,1}$ until r_0^t.
      $C_{i,2} :=$ Decryption of $C_{i,2}$ until r_0^t.
    end
    $\mathcal{X}_1 :=$ PartialDecryption$(k, r, C_{i,1}, t)$
    $\mathcal{X}_2 :=$ PartialDecryption$(k, r, C_{i,2}, t)$
    if $\mathcal{X}_1 \oplus \mathcal{X}_2 = e_{\textit{output difference}}$ **then**
      $T_{k,r} := T_{k,r} + 1$
    end
  end
end
return $\{(k, r) \mid T_{k,r}$ is maximum$\}$

Algorithm 2: One round partial decryption using candidate values k and r for a block ct of length m. The t is the target round

PartialDecryption(k, r, ct, t)
$ct := ct \oplus k$
if $t \leq 2$ **then**
  $ct := ct -_m r$
  $ct := ct \ggg 2$
else
  if $t = 3$ **then**
    $ct := ct \ggg 2$
    $ct := ct -_m r$
  end
end
$ct := ct^{-1}$
return ct

$r = r_7r_6r_5r_4r_3r_2r_1r_0$ and $r +_m 128 = (\neg r_7)r_6r_5r_4r_3r_2r_1r_0$. We have the following:

$$\begin{aligned}&(y -_m r)\\ &= y_7y_6y_5y_4y_3y_2y_1y_0 -_m r_7r_6r_5r_4r_3r_2r_1r_0\end{aligned} \tag{8}$$

$$\begin{aligned}&(y' -_m (r +_m 128)\\ &= (\neg y_7)y_6y_5y_4y_3y_2y_1y_0 -_m (\neg r_7)r_6r_5r_4r_3r_2r_1r_0\end{aligned} \tag{9}$$

Since the most significant bit of both operands is complemented, the result of the modular subtraction will be the same. In both cases, the number of candidate pairs is 2. □

TABLE II
DIFFERENTIAL CHARACTERISTICS USED IN THE ATTACK. EACH COLUMN SHOWS THE POSITION OF THE DIFFERENCE IN THE CHARACTERISTIC. RE(t) COLUMNS SHOW THE POSITION OF THE DIFFERENCE AFTER BITREORDERING(t). $\Pr_{Re(t)}$ SHOWS THE PROBABILITY OF THE CHARACTERSTIC AFTER BITREORDERING(t). THE PROBABILITIES ARE IN THE BASE 2 LOGARITHM OF THE CHARACTERISTIC PROBABILITY. THE ELEMENTS x ARE NOT IMPORTANT BECAUSE THEY ARE NOT USED IN THE ATTACK.

Input	Re(0)	Re(1)	Re(2)	$\Pr_{Re(0)}$	$\Pr_{Re(1)}$	$\Pr_{Re(2)}$
22	17	11	16	-1.1	-6.3	-11.1
15	8	18	13	-0.28	-6.5	-10.9
19	19	0	2	-0.96	-10	-11.8
4	0	x	x	-0.29	x	x

2) Phase 2: Recovering r_0^2 and r_1^1.: Recovering r_0^2 and r_1^1 is simpler since we already recovered k_3, k_2, r_1^2 and we can partially decrypt any ciphertext for one round. Then for every $r_0^2[j]$ and every $k_2[j] + (r_1^1[j] \lll 2)$ we partially decrypt the second round until the XOR addition with r_0^1 and compare the intermediate difference with the expected difference from the corresponding characteristic. The attack starts from the SBox transformations from round 2 until the xor addition of r_0^2. We define the function $F_2 : \{0,1\}^{24} \to \{0,1\}^{24}$ as follow.

$$F_2(X) = P(((S(X) + r_1^1) \lll 2) +_m k_2) \oplus r_0^2 \tag{10}$$

Where P and S represent the BitReordering and SBox function respectively. Since xor and the permutation are both linear and commutative, we can swap their positions. Additionally, the permutation is known, so we can exclude it from the equation because the state after the permutation is known. Therefore, we have

$$F_2(X) = ((S(X) + r_1^1) \lll 2) +_m k_2 \oplus r_0^2. \tag{11}$$

We underline that aim at recovering the value of $k_2 + (r_1^1 \ggg 2)$ instead of the exact values of k_2 and r_1. In fact, knowing only the sum is enough to encrypt/decrypt messages. However, this value can not be recovered directly, because the rotation is applied over the state after the SBox transformations and the modular addition of the randomizer. To overcome this, we use the rotational property for modular addition and can say that

$$(S(X) + r_1^1) \lll 2 = (S(X) \lll 2) +_m (r_1^1 \lll 2) \tag{12}$$

holds with probability 0.4 for rotation constant $l = 2$ and size of the word $n = 8$ bits (see Equation 4). Then

$$F_2(X) = (S(X) \lll 2) +_m (r_1^1 \lll 2) +_m k_2 \oplus r_0^2 \tag{13}$$

holds with probability $p = 0.4 = 2^{-1.32}$. Therefore our characteristic will hold with probability $2^{-7} \times 2^{-1.32} = 2^{-8.32}$. The partial decryption function associated to Algorithm 2 is

$$F^{-1}(Y) = S^{-1}(((Y \oplus r_0^2 -_m k_2 -_m (r_1^1 \ggg 2)) \ggg 2) \tag{14}$$

where Y represents the ciphertext after 2 rounds, recovered by the previous phase. Then we guess the value of $k_2 +_m (r_1^1 \ggg 2)$ and as we already recovered k_2 in Section III-B1, we can recover r_1^1.

3) Phase 3: Recovering r_0^1, r_1^0 and k_1.: We apply the same attack as in Section III-B2, but this time over the first round. The function we partially decrypt over is defined as follow.

$$F_3(X) = (S(X) \lll 2) + (r_1^0 \lll 2) +_m k_1 \oplus r_0^1 \tag{15}$$

and the inverse is

$$\begin{aligned}&F_3^{-1}(Y)\\ &= S^{-1}(Y \oplus r_0^1 -_m k_-(r_1^0 \ggg 2))\end{aligned} \tag{16}$$

The algorithm of recovering r_0^1 and $r_1^0 \ggg 2 +_m k_1$ is described in the Algorithm 1.

4) Phase 4: Recovery of r_0^0 and k_0.: Recovering r_0^0 and k_0 is straightforward since we can decrypt any ciphertext for three rounds. We can guess the values for $r_0^0[j]$ and $k_0[j]$ and verify the guess using 2^{16} different pairs. Let P be the first permutation BitReordering 0, for each pair 8-bit values (X, Y), there are 2^8 possible values of r_0^0 and k_0 such that $P(X +_m k_0) \oplus r_0^0 = Y$.

C. Complexity of the attack

The data complexity is computed as the number of times we call the encryption/decryption function. In the first phase of the attack, the probability of each characteristic is lower bounded by $p = 2^{-12}$ hence exploiting each differential characteristic requires encrypting at least $\frac{1}{p}$ pairs of chosen plaintexts. In our attack, we use $\frac{4}{p}$ pairs of chosen plaintext to guarantee a successful attack. The partial decryption in the last round involves one SBox out of the total 9 SBoxes that are computed in one encryption/decryption, which can be estimated to a total of $2^{15} \cdot \frac{1}{9}$ decryptions. Furthermore, we search the whole space of two 8-bit values to compute the respective key and randomizer values. Therefore the total complexity of the first phase is $3 \cdot 2^{16} \cdot 2^{15} \cdot \frac{1}{9} \approx 2^{29.4}$. The second phase has less complexity since the used characteristics are over 1 round and have a larger probability. The complexity of the second phase is $2^{27.4}$ and the complexity of the third phase is $2^{21.4}$. The total complexity of the attack is $2^{29.7}$.

The complexity of the attack depends on the probability of the characteristics, which depends on the GDDT of the SBoxes. Thus, we can say that all versions of the K-Cipher that have the same size SBoxes have the same theoretical

complexity. However, in practice, the execution of the attack will have different measured times. This happens because the time required for one run of the algorithm is proportional to the number of SBoxes in one encryption. Hence, the smaller the state, the faster the attack. One way to compare the practical time is to look at the complexity as the number of SBox look-ups instead of the number of encryptions. For example, a 24-bit state with an 8-bit SBox would take $2^{29.4} \cdot 9$ table look-ups, while a 64-bit state with an 8-bit SBox will require $2^{29.4} \cdot 24$ table look-ups. We expect the second version to be around 2.5 times slower.

D. Experimental verification of the attack

We implemented our attack in C++ and run it on a 5.13.0-35-generic $x86\_64$ Linux machine with 32 GiB of memory and Intel Core i7 processor. Our implementation terminates after 1812s and recovers all the secret data. The implementation of the attack is available on Github[1].

E. Other parameters for K-Cipher

The main reason for the choice of parameters in this paper is that K-Cipher is mostly intended to be used for small states [1]. However, the same attack can be applied to any other choice of parameters. There are three main parameters that affect the overall security of the system: The state size, the block size, and the number of rounds. The block size determines the order of the underlying binary field and for larger orders, the probability of the differential characteristics is smaller and the DDTs are different. For the same block size, the larger state does not increase the complexity of the attack but increases the running time of the algorithm. The larger number of rounds results in characteristics with smaller probability and hence increases the complexity of the attack.

The other main difference for states of size $n > 32$, is that the round keys are derived from a key schedule algorithm and they are not independent. The dependence of round keys needs two adjustments in the attack we described in this paper. First, after recovering k_3, we can recover all other round keys using the Algorithm 3. Second, we can not choose any member of an equivalence class of the candidates and we need to guess the correct one. This can be done efficiently using exhaustive search and verification of each guess.

IV. Conclusion

The main motivation of designing K-Cipher was to have a low latency cipher that is efficient in different use cases. The original design with deterministic SBox layers was claimed to be vulnerable to differential cryptanalysis. Therefore, to improve its security, the designers suggested increasing the key length and using additional randomness in the SBox layer. The attack presented here showed that not only the increase of the key length does not result in a secure cipher but it also creates duplicate keys which is not a desirable property for a block cipher. Apart from the existence of the GDDTs, the attack exploited the low number of rounds of the cipher. Although the number of rounds improves the latency, it jeopardizes security.

[1] https://github.com/KULeuven-COSIC/K-Cipher.

Algorithm 3: Recovering all the round keys. ReverseBitReordering functions are reversing the permutations specified in K-Cipher paper. The subtraction $k_i -_m c_j$ is a modular subtraction.

Input: k_3
$k_2 = \text{ReverseBitReordering}(k_3, 3)$
$k_1 = \text{ReverseBitReordering}(k_2, 7)$
$k_1 = S^{-1}(k_1)$
$k_1 = \text{ReverseBitReordering}(k_1, 6)$
$k_1 = k_1 -_m c_2$
$k_0 = \text{ReverseBitReordering}(k_1, 5)$
$k_0 = S^{-1}(k_0)$
$k_0 = \text{ReverseBitReordering}(k_0, 4)$
$k_0 = k_0 -_m c_1$
return k

References

[1] M. Kounavis, S. Deutsch, S. Ghosh, and D. Durham, "K-cipher: A low latency, bit length parameterizable cipher," in *2020 IEEE Symposium on Computers and Communications (ISCC)*, 2020, pp. 1–7.

[2] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin, "PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract," in *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, ser. Lecture Notes in Computer Science, X. Wang and K. Sako, Eds., vol. 7658. Springer, 2012, pp. 208–225. [Online]. Available: https://doi.org/10.1007/978-3-642-34961-4_14

[3] D. Bozilov, M. Eichlseder, M. Knezevic, B. Lambin, G. Leander, T. Moos, V. Nikov, S. Rasoolzadeh, Y. Todo, and F. Wiemer, "Princev2 - more security for (almost) no overhead," in *Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*, ser. Lecture Notes in Computer Science, O. Dunkelman, M. J. J. Jr., and C. O'Flynn, Eds., vol. 12804. Springer, 2020, pp. 483–511. [Online]. Available: https://doi.org/10.1007/978-3-030-81652-0_19

[4] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim, "The SKINNY family of block ciphers and its low-latency variant MANTIS," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. Robshaw and J. Katz, Eds., vol. 9815. Springer, 2016, pp. 123–153. [Online]. Available: https://doi.org/10.1007/978-3-662-53008-5_5

[5] R. Avanzi, "The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes," *IACR Trans. Symmetric Cryptol.*, vol. 2017, no. 1, pp. 4–44, 2017. [Online]. Available: https://doi.org/10.13154/tosc.v2017.i1.4-44

[6] G. Leander, T. Moos, A. Moradi, and S. Rasoolzadeh, "The SPEEDY family of block ciphers engineering an ultra low-latency cipher from gate level for secure processor architectures," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 4, pp. 510–545, 2021. [Online]. Available: https://doi.org/10.46586/tches.v2021.i4.510-545

[7] M. LeMay, J. Rakshit, S. Deutsch, D. M. Durham, S. Ghosh, A. Nori, J. Gaur, A. Weiler, S. Sultana, K. Grewal, and S. Subramoney, "Cryptographic capability computing," in *MICRO '21: 54th Annual IEEE/ACM International Symposium on Microarchitecture, Virtual Event, Greece, October 18-22, 2021*. ACM, 2021, pp. 253–267. [Online]. Available: https://doi.org/10.1145/3466752.3480076

[8] NIST, "FIPS-46: Data Encryption Standard (DES)." 1979, http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf. [Online]. Available: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[9] L. R. Knudsen, "Truncated and Higher Order Differentials," in *FSE*, ser. Lecture Notes in Computer Science, vol. 1008. Springer, 1994, pp. 196–211.

[10] D. A. Wagner, "The Boomerang Attack," in *FSE*, ser. Lecture Notes in Computer Science, vol. 1636. Springer, 1999, pp. 156–170.

[11] E. Biham, A. Biryukov, and A. Shamir, "Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, vol. 1592. Springer, 1999, pp. 12–23.

[12] M. Daum, "Cryptanalysis of hash functions of the md4-family," 2005.

[13] M. E. Kounavis, S. Deutsch, S. Ghosh, and D. Durham, "K-cipher: A low latency, bit length parameterizable cipher," *IACR Cryptol. ePrint Arch.*, p. 30, 2020. [Online]. Available: https://eprint.iacr.org/2020/030