JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 34, 733-748 (2018)
DOI: 10.6688/JISE.201805_34(3).0010

Cryptanalysis of the Round-Reduced Kupyna

JIAN ZOU[1,2] AND LE DONG[3]
[1]*Department of Mathematics and Computer Science*
[2]*Key Lab of Information Security of Network Systems*
Fuzhou University
Fuzhou, 350108 P.R. China
[3]*Henan Engineering Laboratory for Big Data Statistical Analysis and Optimal Control*
Henan Normal University
Xinxiang, 453007 P.R. China
E-mail: {fzuzoujian15; dongle127}@163.com

Kupyna was approved as the new Ukrainian hash standard in 2015. In this paper, we show several pseudo-preimage attacks and collision attacks on Kupyna. Due to the wide-pipe design, it is hard to construct pseudo-preimage attacks on Kupyna. Combining the meet-in-the-middle attack with the guess-and-determine technique, we propose some pseudo-preimage attacks on the compression function for 5-round Kupyna-256 and 7-round Kupyna-512. The complexities of these two pseudo-preimage attacks are $2^{229.5}$ (for 5-round Kupyna-256) and 2^{499} (for 7-round Kupyna-512) respectively. Regarding the collision attack, we can not only construct a collision attack on the 7-round Kupyna-512 compression function with a complexity of $2^{159.3}$, but also construct a collision attack on the 5-round Kupyna-512 hash function with a complexity of 2^{240}.

Keywords: Kupyna, pseudo-preimage, collision, rebound attack, meet-in-the-middle

1. INTRODUCTION

Cryptographic hash functions are playing important roles in the modern cryptography. In general, hash function must satisfy three security requirements: preimage resistance, second preimage resistance and collision resistance.

Regarding the preimage attack, the meet-in-the-middle (MitM) attack proposed by Aoki and Sasaki [1] was a generic method to construct preimage attacks against hash function. From then on, the MitM attack has been applied to obtain preimages for many hash functions such as Tiger [2], RIPEMD [3] and SHA-2 [2]. At FSE 2012, Wu *et al.* [10] proposed the pseudo-preimage attacks on Groestl by combining the MitM attacks with the technique proposed by Wagner [13]. Then Zou *et al.* [12] proposed some improved pseudo-preimage attacks on Groestl by using the guess-and-determine MitM attack.

At FSE 2009, Mendel *et al.* [7] proposed the rebound attack to construct some collision attacks on AES-based designs. From then on, many techniques were proposed to improve the rebound attack such as start-from-the-middle [6], and Super-Sbox [5]. At FSE 2014, Mendel *et al.* [8] used the rebound attack to construct the collision attacks on 5-round Groestl hash function.

In 2015, Kupyna [9] was selected as the new Ukrainian standard. It employs a similar design as Groestl. The amounts of cryptanalytic results on Kupyna are rare. We summarize the related works and our attacks in Table 1. In this paper, we present the first

Received December 8, 2016; revised March 14, 2017; accepted April 16, 2017.
Communicated by Xuejia Lai.

Table 1. Comparison to previous works.

Algorithm	Target	Attack Type	Rounds	Time	Source
Kupyna-256	CF	Semi-free-start Collision	6	2^{70}	[4]
	CF	Semi-free-start Collision	7	$2^{125.8}$	[4]
	HF	Collision	5	2^{120}	[4,11]
	OT	Preimage	6	2^{240}	[11]
	HF	Pseudo-Preimage	6	$2^{250.33}$	[11]
	CF	Pseudo-Preimage	5	$2^{229.5}$	Section 4.2
Kupyna-512	OT	Preimage	8	2^{472}	[11]
	HF	Pseudo-Preimage	8	$2^{498.33}$	[11]
	CF	Pseudo-Preimage	7	2^{499}	Section 4.3
	CF	Semi-free-start Collision	7	$2^{159.3}$	Section 5
	HF	Collision	5	2^{240}	Section 6

HF: Hash Function, CF: Compression Function, OT: Output Transformation.

pseudo-preimage attacks on the compression function of Kupyna. Due to the wide-pipe design, it is difficult to construct pseudo-preimage attacks on Kupyna. Nevertheless, we find out that the pseudo-preimage attack on Kupyna could be turned into a k-sum problem. And it's possible to break the birthday bound by using the guess-and-determine MitM attack and the algorithm in [13] for solving the k-sum problem. However, the algorithm in [13] would require a large amount of space. For example, Wagner in [13] showed how to find a solution to the 4-sum problem with $2^{n/3}$ time and $2^{n/3}$ space. With the above time-memory trade-off techniques, we propose the first pseudo-preimage attacks on the compression function for 5-round Kupyna-256 and 7-round Kupyna-512. Since Kupyna adopts a similar design as Groestl, we can construct pseudo-preimage attacks on Groestl in a similar way. Our preimage attacks show that the modular constant addition operation provides some additional security against the MitM attack on the underlying permutations of Kupyna, because it would cause confusion in each column via the carry.

Since Kupyna adopts the AES design, we can utilize the rebound attack to construct a semi-free-start collision attack on the compression function for 7-round Kupyna-512. In [8], Mendel *et al*. showed a method to construct the collision attacks on 5-round Groestl hash function. However, they did not show the details or the complexity of their collision attack on 5-round Groestl-512. In this paper, we will show the details and the complexity of our collision attack on 5-round Kupyna-512 hash function.

The rest of the paper is organized as follows. A short description of Kupyna is given in Section 2. In Section 3, we show some techniques that would be used in our pseudo-preimage attacks on Kupyna. Then we show some pseudo-preimage attacks on the compression function of Kupyna in Section 4. In Sections 5 and 6, we present the collision attacks on Kupyna-512. Section 7 concludes the paper.

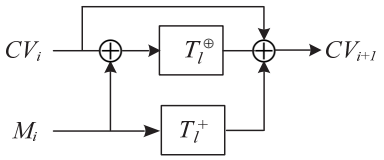
2. DESCRIPTION OF KUPYNA

Kupyna employs the wide-pipe design. The maximum length of the input message is limited to $2^{96}-1$ bits. The input message M is padded into a multiple of l bits by the padding procedure. If $8 \le n \le 256$, $l = 512$, else $l = 1024$. Then we put another 96 bits in-

cluding the length of $|M|$ at the end of M. The padded message M^* is divided into l bits blocks M_i ($i = 0, 1, \ldots, k-1$). An output hash h can be generated as follows:

$$CV_0 \leftarrow IV;\ CV_{i+1} \leftarrow CF_l(CV_i, M_i) \text{ for } i = 0, 1, \ldots, k-1, \text{ and } h = Trunc_n(T_l^{\oplus}(CV_k) \oplus CV_k).$$

The $Trunc_n(x)$ returns the most significant n bits of x, and $CF_l(CV_i, M_i)$ is the compression function of Kupyna. It contains two permutations $T_l^{\oplus}$ and T_l^{+}, and computes as follows (see in Fig. 1 (a)): $CF_l(CV_i, M_i) = T_l^{\oplus}(CV_i \oplus M_i) \oplus T_l^{+}(M_i) \oplus CV_i$.



(a) Compression function.

$g_{0,0}$	$g_{0,1}$	$g_{0,2}$	$g_{0,3}$	$g_{0,4}$	$g_{0,5}$	$g_{0,6}$	$g_{0,7}$
$g_{1,0}$	$g_{1,1}$	$g_{1,2}$	$g_{1,3}$	$g_{1,4}$	$g_{1,5}$	$g_{1,6}$	$g_{1,7}$
$g_{2,0}$	$g_{2,1}$	$g_{2,2}$	$g_{2,3}$	$g_{2,4}$	$g_{2,5}$	$g_{2,6}$	$g_{2,7}$
$g_{3,0}$	$g_{3,1}$	$g_{3,2}$	$g_{3,3}$	$g_{3,4}$	$g_{3,5}$	$g_{3,6}$	$g_{3,7}$
$g_{4,0}$	$g_{4,1}$	$g_{4,2}$	$g_{4,3}$	$g_{4,4}$	$g_{4,5}$	$g_{4,6}$	$g_{4,7}$
$g_{5,0}$	$g_{5,1}$	$g_{5,2}$	$g_{5,3}$	$g_{5,4}$	$g_{5,5}$	$g_{5,6}$	$g_{5,7}$
$g_{6,0}$	$g_{6,1}$	$g_{6,2}$	$g_{6,3}$	$g_{6,4}$	$g_{6,5}$	$g_{6,6}$	$g_{6,7}$
$g_{7,0}$	$g_{7,1}$	$g_{7,2}$	$g_{7,3}$	$g_{7,4}$	$g_{7,5}$	$g_{7,6}$	$g_{7,7}$

(b) Byte positions for Kupyna-256.

Fig. 1. Compression function of Kupyna and the byte positions for Kupyna-256.

$T_l^{\oplus}$ and T_l^{+} are AES-based permutations, which contain 8×8 and 8×16 sized states for Kupyna-256 and Kupyna-512 respectively. The state for Kupyna-256 (see in Fig. 1 (b)) is denoted as $G = (g_{i,j})$, $g_{i,j} \in GF(2^8)$, where $i = 0, 1, \ldots, 7, j = 0, 1, \ldots, 7$. The $G' = (g_{i,j})$ ($i = 0, 1, \ldots, 15, j = 0, 1, \ldots, 7$) for Kupyna-512 can be constructed in a similar way. Kupyna-256 adopts 10-round $T_{512}^{\oplus}$ and T_{512}^{+}, while Kupyna-512 employs 14-round $T_{1024}^{\oplus}$ and T_{1024}^{+}. $T_l^{\oplus}$ and T_l^{+} adopt four operations, which are defined as follows:

– **AddConstant:** This operation is different for $T_l^{\oplus}$ and T_l^{+}. T_l^{+} adopts column wise modular addition mod 2^{64}, while $T_l^{\oplus}$ uses bitwise xor $\oplus$.
– **SubBytes:** This operation applies an S-box to each cell of the state.
– **RotateBytes:** The RotateBytes transformation cyclically rotates the cells of the ith row rightwards by shift vector (define later).
– **MixColumns:** Each column of the matrix should be multiplied by an MDS matrix in the MixColumns operation.

We use AC, SB, RB and MC to denote the above four operations for short. Since the AC operations for $T_l^{\oplus}$ and T_l^{+} are different, we denote AC_1 and AC_2 for $T_l^{\oplus}$ and T_l^{+}, respectively. The AC_1 is expressed as $w_j^{(r)} = ((j << 4)r, 00, 00, 00, 00, 00, 00, 00)^T$ for round r and column j, while the AC_2 operation for column j is expressed as $\zeta_j^{(r)} = (F3, F0, F0, F0, F0, F0, F0, (7-j) << 4)^T$. The shift vectors used in $T_l^{\oplus}$ and T_l^{+} are defined as below, $T_{512}^{\oplus}$ and T_{512}^{+} in Kupyna-256 uses (0,1,2,3,4,5,6,7), while $T_{1024}^{\oplus}$ and T_{1024}^{+} in Kupyna-512 uses (0,1, 2,3,4,5,6,11). For a detailed description of Kupyna, we refer to the original paper [9].

3. PREVIOUS WORKS

In this section, we present some techniques used in our preimage attacks on Kupyna.

3.1 The Algorithm to Solve the *k*-sum Problem

In CRYPTO 2002, Wagner [13] proposed an algorithm with subexponential running time to solve the *k*-sum problem. For example, he showed the 4-sum problem could be solved with $2^{n/3}$ time and memory. This algorithm could be seen as a time-memory trade-off algorithm, which would reduce the time complexity by increasing the space requirement. Suppose the adversaries were given four lists L_1, L_2, L_3, L_4, and they wanted to find values $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$, where $x_i \in L_i$. The attack process of the 4-sum problem ($x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$) could be summarized as follows. Firstly, they constructed two new lists L_{12} and L_{34}, where L_{12} contained values of $x_1 \oplus x_2$ such that $Trunc_{l_1}(x_1 \oplus x_2) = 0$ and L_{34} contained values of $x_3 \oplus x_4$ satisfying $Trunc_{l_1}(x_3 \oplus x_4) = 0$. Secondly, they searched for some matches between L_{12} and L_{34}. The match would satisfy $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$, which meant the adversaries had found a solution to the 4-sum problem.

As shown in [13], these were several efficient methods for constructing the lists L_{12} and L_{34}, such as the merge-join method and the hash-join method. The hash-join method stored the lists L_1 and L_2 in hash tables, and then checked whether these was one element satisfying $Trunc_{l_1}(x_1 \oplus x_2) = 0$ in L_1 by scanning through L_2. The hash-join method was very efficient when memory was plentiful. It only required $|L_1|+|L_2|$ computation and $\min(|L_1|, |L_2|)$ space, while the merge-join method needed $O(n_2 \log n_2)$ time ($n_2 = \max(|L_1|, |L_2|)$). Then the complexity of the above algorithm could be calculated as follows: the probability of $Trunc_{l_1}(x_1 \oplus x_2) = 0$ was $1/2^{l_1}$ when x_1 and x_2 were chosen uniformly at random. Thus, the size of L_{12} was $2^{2l_1} / 2^{l_1} = 2^{l_1}$. Similarly, the size of L_{34} was also 2^{l_1}. The expected number of a match between L_{12} and L_{34} was about $|L_{12}| \times |L_{34}| / 2^{n-l_1}$, which was at least 1 when $l_1 \geq n/3$. This meant the 4-sum problem could be implemented with $2^{n/3}$ time and memory. The other *k*-sum problem can be solved in a similar way.

3.2 The Pseudo-Preimage Attacks on Groestl

In FSE 2012, Wu *et al*. [10] proposed the first pseudo-preimage attacks on Groestl. Suppose the hash output was *n*-bit and the state size was 2*n*-bit. Due to the wide-pipe design of Groestl, it seemed impossible to construct a pseudo-preimage attack with a complexity 2^{n_1}, where $n_1 < n$. Assume $X = CF(CV, M)$, then X was the preimage of the output transformation. Let $H' = CV \oplus M$, Wu *et al*. obtained the equation of $(P(H') \oplus H') \oplus (Q(M) \oplus M) \oplus X = 0$, which meant their pseudo-preimage attacks could be turned into a *k*-sum problem (see in Fig. 2). In order to reduce the time complexities of their pseudo-preimage attacks, Wu *et al*. implemented their attacks by combining the MitM attack with the algorithm in [13] for solving the *k*-sum problem. Especially, Wu *et al*. used the hash-join method in their pseudo-preimage attacks to reduce the time complexity. However, the algorithm in [13] would need a large amount of memory. Thus, the pseudo-preimage attacks of Wu *et al*. needed a lot of space resources. Their attack process in [10] can be described with four parameters x_1, x_2, x_3 and b as follows:

Step 1: Obtain 2^{x_1} preimages of X and store them in a hash table L_{h1}.
Step 2: Calculate $2^{x_3} H'$ such that the leftmost b-bit of $P(H') \oplus H'$ are zero. Store these values of $P(H') \oplus H'$ in a hash table L_{h2}.
Step 3: Choose 2^{x_2} random M with the correct padding and compute $Q(M) \oplus M$. Check

whether there exists an entry in L_{h1} that matches the leftmost b bits of $Q(M)\oplus M$. These are $2^{x_1+x_2-b}$ partial matches of $Q(M)\oplus M\oplus X$ are expected to remain.
Step 4: For all $2^{x_1+x_2-b}$ $Q(M)\oplus M\oplus X$ left in Step 3, check whether there exists an element in L_{h2} that satisfying $(P(H')\oplus H')\oplus(Q(M)\oplus M)\oplus X = 0$. Once an adversary fined a full match, they obtain a pseudo-preimage of Groestl.

Assume that an adversary needs $2^{C_1(2n,n)}$ computations to obtain a preimage of X and $2^{C_2(2n,b)}$ computations to find a chosen b-bit partial preimage of $P(X)\oplus X$. Since Wu *et al*. used the hash-join method, the complexity of their attack could be computed as follows: an adversary needs $2^{x_1+C_1(2n,n)}$ computations and 2^{x_1} memory to build the table L_{h1} in Step 1. Then the adversary takes $2^{x_3+C_2(2n,b)}$ computations and 2^{x_3} memory to build the table L_{h2} in Step 2. In order to check the partial match in table L_{h1}, they needs 2^{x_2} Q calls to calculate $Q(M)\oplus M$ in Step 3, which is equivalent to 2^{x_2-1} compression function calls. In Step 4, the adversary needs $2^{x_1+x_2-b}$ table look-ups to check the final $2n-b$ bits match for the left $2^{x_1+x_2-b}$ candidates, which can be equivalently seen as $2^{x_1+x_2-b}\cdot C_{TL}$ compression function calls. Here C_{TL} is the complexity of one table lookup, where unit one is one compression function call. The overall complexity to compute a pseudo-preimage of Groestl is:

$2^{x_1+C_1(2n,n)} + 2^{x_3+C_2(2n,b)} + 2^{x_2-1} + 2^{x_1+x_2-b}2^{x_3+C_2(2n,b)}\cdot C_{TL}$, with memory requirement of $2^{x_1} + 2^{x_3}$.

In [10], Wu *et al*. showed several pseudo-preimage attacks on Groestl. For example, they needed $2^{244.85}$ computations and $2^{230.13}$ memories to construct the pseudo-preimage attack on 5-round Groestl-256. And it took $2^{507.32}$ computations and $2^{507.00}$ memories to construct a pseudo-preimage attack on 8-round Groestl −512. Since the design of Kupyna is similar to Groestl, we will adopt the same techniques used in [10] to attack Kupyna. Note that our pseudo-preimage attacks on Kupyna also need a lot of memory resources.

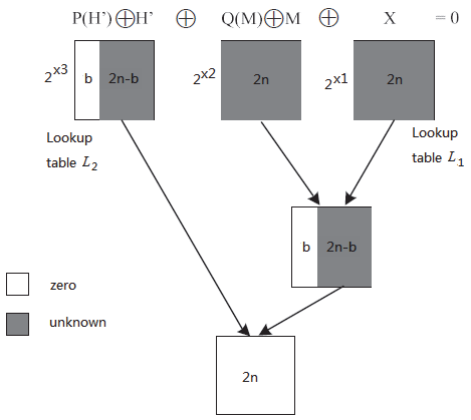


Fig. 2. Outline for Wu's attack on Groestl.

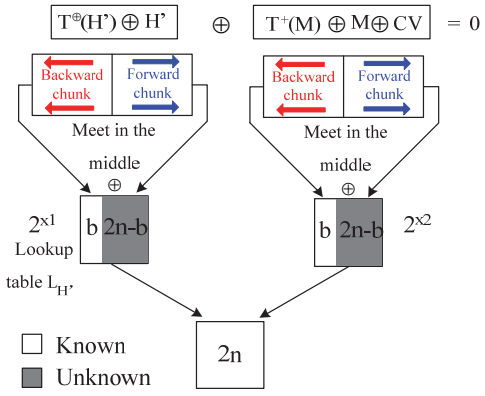


Fig. 3. Outline for our attack on Kupyna.

4. PSEUDO-PREIMAGE ATTACKS ON KUPYNA

In this section, we will present some pseudo-preimage attacks on Kupyna. The $CF_l(CV_i, M_i)$ can be rewritten as $CF_l(CV_i, M_i) = T_l^{\oplus}(H_i')\oplus H_i'\oplus T_l^{+}(M_i)\oplus M_i$, if we let $H_i' = CV_i\oplus M_i$. We will use this important property in our pseudo-preimage attacks.

4.1 Outline the Pseudo-Preimage Attacks on the Compression Function of Kupyna

In this section, we show how to construct the pseudo-preimage attacks on the compression function of Kupyna. Our pseudo-preimage attacks are similar to Wu's attacks [10], which also need a lot of memory resources. In detail, we find out that our pseudo-preimage attacks (in Fig. 3) can be turned into a variant 4-sum problem. In order to reduce the time complexity of our pseudo-preimage attacks, we should adopt the similar techniques used in [10, 13], such as the MitM attack and the hash-join method. As shown in Fig. 3, we want to find a pair of (H', M) satisfying the equation $T_l^{\oplus}(H')\oplus H'\oplus T_l^{+}(M)\oplus M = CV$, then (H', M) is a pseudo-preimage. Our pseudo-preimage attack with parameters x_1, x_2 and b can be described as follows.

Step 1: Obtain $2^{x_1}H'$ such that the chosen b-bit of $Trunc_b'(T_l^{\oplus}(H')\oplus H')$ are all zero. The 2^{x_1} value of $T_l^{\oplus}(H')\oplus H'$ are stored in a hash table $L_{H'}$.
Step 2: Find 2^{x_2} M such that $Trunc_b'(T_l^{+}(M)\oplus M) = Trunc_b'(CV)$. For each of the resulting 2^{x_2} values of $T_l^{\oplus}(M)\oplus M$, check whether there exists an element in $L_{H'}$ satisfying $Trunc_{2n-b}'(T_l^{\oplus}(H')\oplus H'\oplus T_l^{+}(M)\oplus M) = Trunc_{2n-b}'(CV)$.

The $Trunc_b'(x)$ function returns the chosen b bits of x. We will adopt the hash-join method (see in Section 3.1) in our pseudo-preimage attacks, which means we store the 2^{x_1} values of $T_l^{\oplus}(H')\oplus H'$ in a hash table. The hash-join method would reduce the time complexity of our pseudo-preimage attacks. Suppose we need $2^{C_1(n)}$ and $2^{C_2(n)}$ computation to find an H' and M in Steps 1 and 2 respectively. Then we need $2^{x_1+C_1(n)}$ computations and 2^{x_1} memories to construct $L_{H'}$ in step 1. In Step 2 we need $2^{x_2+C_2(n)}$ computations to find 2^{x_2} M. For each element of the 2^{x_2} values of $T_l^{+}(M)\oplus M$, we should check whether there exists an entry in $L_{H'}$ that matches the final $2n-b$ bits. The complexity to obtain a pseudo-preimage of the compression function of Kupyna is $2^{x_1+C_1(n)}+2^{x_2+C_2(n)}$. The condition of $2^{x_1+x_2+2n+b} \geq 1$ should be satisfied so as to find one final match.

4.2 Pseudo-Preimage Attack on 5-round Kupyna-256 Compression Function

In this section, we present a pseudo-preimage attack on the compression function for 5-round Kupyna-256. In detail, we will show how to use the MitM attacks to find 2^{x_1} and 2^{x_2} candidates for $T_{512}^{\oplus}(H')\oplus H'$ and $T_{512}^{+}(M)\oplus M$ respectively.

MitM Attack on $T_{512}^{\oplus}(H')\oplus H'$ In Fig. 4 (a), we use the red/blue color to denote neutral messages, which are independent from each other. The gray bytes are constants. We utilize the white color to denote the bytes whose values are unknown. Some parameters for the MitM attack are needed: the size of the matching point m, the freedom degrees D_r (for red bytes) and D_b (for blue bytes). In order to achieve a better complexity, we define

other parameters: freedom degrees for red and blue bytes (d_r, d_b) used in our attack and the size of the chosen matching point b_m. Note that (d_r, d_b, b_m) and (D_r, D_b, m) can be different. Our MitM attack on $T^{\oplus}_{512}(H')\oplus H'$ can be described as follows:

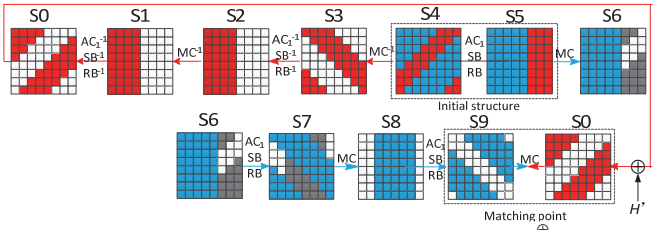
Step 1: Choose the constants in the initial structure randomly.

Step 2: Compute backward for 2^{dr} of the red bytes, and obtain the values at the matching point. Store the results in a list L_r.

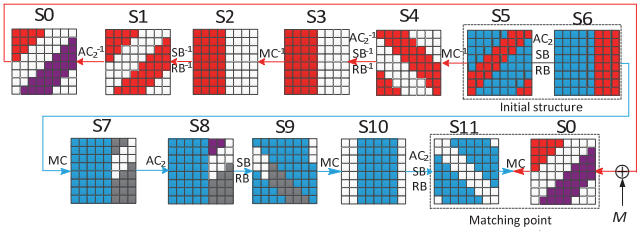
Step 3: Compute forward for 2^{d_b} of the blue bytes, and obtain the values at the matching point. Check if there exists an entry in L_r that satisfies the values of the chosen b-bit of $T^{\oplus}_{512}(H')\oplus H'$ are all zero at the matching point. There are 2^{dr+d_b-bm} partial matches left.

The initial structure is the starting point of our MitM attack. As shown in Fig. 4 (a), the neutral bytes for the forward chunk are 24 bytes $S_5[g_{i,5}, g_{i,6}, g_{i,7}]$ (for $0 \le i \le 7$). We can choose 9 bytes ($= 2^{72}$) values of $S_5[g_{i,5}, g_{i,6}, g_{i,7}]$ (for $0 \le i \le 2$) freely. The values of the left 15 bytes of $S_5[g_{i,5}, g_{i,6}, g_{i,7}]$ (for $3 \le i \le 7$) can be computed as follows. Firstly, choose the 15 bytes of the constant bytes at S_6 (in gray color) randomly. Secondly, compute the 15 bytes values of $S_5[g_{i,5}, g_{i,6}, g_{i,7}]$ (for $3 \le i \le 7$) so that the chosen 15 bytes of the constant bytes at S_6 can be obtained through the Inverse MC operation (or call MC^{-1}), for all 2^{72} values of $S_5[g_{i,5}, g_{i,6}, g_{i,7}]$ (for $0 \le i \le 2$). Note that this calculation can always be achieved because we have 15 free variables to control 15 bytes for a system of linear equations. For all 2^{72} values of the neutral bytes for the forward chunk, we can compute these state values (in blue color) at S_9. Similarly, the neutral bytes for the backward chunk are 40 bytes of $S_5[g_{i,0}, g_{i,1}, g_{i,2}, g_{i,3}, g_{i,4}]$ (for $0 \le i \le 7$), and we can choose 8 bytes freely. Obviously, the neutral bytes for the forward and backward chunk are independent from each other. To sum up, we can obtain $D_r = 64$ bits and $D_b = 72$ bits.

The matching point is also an important part of our MitM attack. We should match the neutral bytes for the forward chunk and the backward chunk through MC operation. Suppose we have collected many values of the partially known states of S_0 and S_9. As shown in Fig. 4 (a), the MC operation can be seen as the linear equations system, which will reduce the number of candidates by a factor of 2^{-8} per column. Thus, $m = 64$ bits.



(a) Chunk separation of 5-round preimage attack on $T^{\oplus}_{512}(H')\oplus H'$.



(b) Chunk separation of 5-round preimage attack on $T^{+}_{512}(M)\oplus M$.

Fig. 4. The MitM attack on 5-round $T^{\oplus}_{512}(H')\oplus H'$ and $T^{+}_{512}(M)\oplus M$.

To reduce the time complexity, we choose $d_r = 64$, $d_b = 64$ and $b_m = 64$ bits in this MitM attack. Because we should find a match between $T^{\oplus}_{512}(H')\oplus H'$ and $T^{+}_{512}(M)\oplus M$, and the best parameters for $T^{+}_{512}(M)\oplus M$ are $d_r = 62$, $d_b = 57$, and $b_m = 64$ bits. Then we can achieve some balance between the two MitM attacks. We can obtain $2^{64}(= 2^{64+64}/2^{64})$ 64-bit partial target preimages with a complexity of 2^{64}. A 64-bit partial target preimage for $T^{\oplus}_{512}(H')\oplus H'$ can be found with a complexity of $2^0(= 2^{64}/2^{64})$.

MitM Attack on $T^{+}_{512}(M)\oplus M$ This MitM attack (in Fig. 4 (b)) is similar to the above attack on $T^{\oplus}_{512}(H')\oplus H'$. However, the AC_2 operation would cause some problems in this attack, because it would cause confusion between the 8 bytes in a column via the carry. Problem 1: the neutral bytes in red and blue would affect each other through the AC_2 in the initial structure. Problem 2: the 22 purple bytes at S_1 and the 3 purple bytes at S_8 may be unknown after the AC_2 operation in Fig. 4 (b). The two problems can be solved by the following methods. First, we let the most significant bit of $S_5[g_{0,4}, g_{0,7}, g_{1,3}, g_{1,6}, g_{2,2}, g_{2,5}, g_{3,1}, g_{3,4}, g_{4,0}, g_{4,3}, g_{5,2}, g_{5,7}, g_{6,1}, g_{6,6}]$ be 1, then carries of these bytes through AC_2 must be 1. In this way, we can solve the Problem 1. As shown in Fig. 4 (b), the freedom of the red and blue bytes are $D_r = 64-7 = 57$ bits and $D_b = 72-7 = 65$ bits, which are lower due to the assignments of the bytes in S_5. Second, the purple bytes at S_0 and S_8 would be unknown after AC_2. However, we can obtain the value of the purple bytes at S_0 and S_8 by guessing the value of the carries through AC_2. Then we should define some new parameters of the guessing of red and blue bits (D_{gr}, D_{gb}) in this attack. Our MitM attack on $T^{+}_{512}(M)\oplus M$ can be described as follows:

Step 1: Let the most significant bit of the chosen bytes be 1. Choose constants randomly.
Step 2: Compute backward for 2^{dr} of the red bytes and 2^{Dgr} of the guessing carry bits, and obtain the values at the matching point. Store the results in a list L'_r.
Step 3: Compute forward for 2^{d_b} of the blue bytes and 2^{Dgb} of the guessing carry bits, and obtain the values at the matching point. Check if there exists an entry in L'_r that matches the equation $Trunc'_b(T^{+}_{l}(M)\oplus M)=Trunc'_b(CV)$ at the matching point. There are $2^{dr+db+Dgr+Dgb-bm}$ partial matches left.
Step 4: Check if the guessed value is right, when a match is found. There are $2^{dr+db-bm}$ valid partial matches left since the probability of the validity is $2^{-Dgr-Dgb}$.

The parameters for our MitM attack on $T^{+}_{512}(M)\oplus M$ are as follows: $d_r = 62$ bits, $d_b = 57$ bits, $D_{gr} = 7$ bits, $D_{gb} = 2$ bits, $b_m = 64$ bits. We can get $2^{55}(= 2^{57+62}/2^{64})$ 64-bit partial target preimages with a complexity of 2^{64}. Then a 64-bit partial target preimage for $T^{+}_{512}(M)\oplus M$ can be found with the complexity of $2^9(= 2^{64}/2^{55})$.

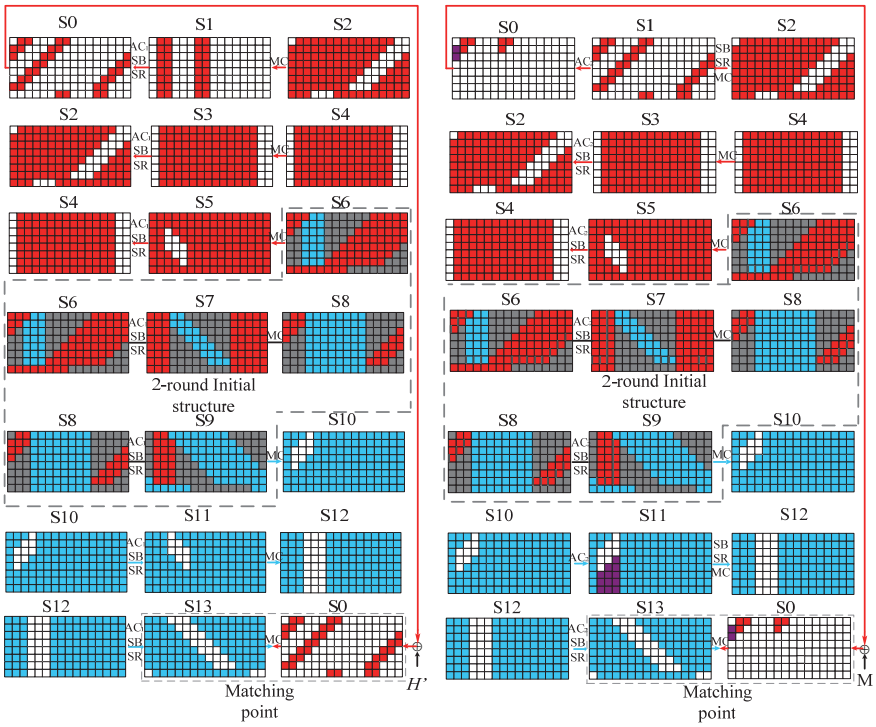
Minimizing the Overall Complexity Our pseudo-preimage attack on 5-round Kupyna-256 compression function can be obtained with a complexity of $2^{x_1}+2^{x_2+9}$, where $x_1 + x_2 \geq 512-64$. As a result, the overall complexity is the lowest ($\approx 2^{229.5}$), when we choose $x_1 = 2^{228.5}$ and $x_2 = 2^{219.5}$. The memory requirement is about $2^{228.5}$.

4.3 Pseudo-Preimage Attack on 7-round Kupyna-512 Compression Function

Our pseudo-preimage attack on the compression function for 7-round Kupyna-512

consists of a 7-round MitM attack on $T^{\oplus}_{1024}(H')\oplus H'$ and a 7-round MitM attack on $T^{+}_{1024}(M)\oplus M$. We can construct a similar 2-round initial structure in [12] for $T^{\oplus}_{1024}(H')\oplus H'$ and $T^{+}_{1024}(M)\oplus M$ (see in Fig. 5).

MitM Attack on $T^{\oplus}_{1024}(H')\oplus H'$ Our MitM attack on $T^{\oplus}_{1024}(H')\oplus H'$ is similar to the MitM attack on $T^{\oplus}_{512}(H')\oplus H'$. We just omit the details of this attack. The best parameters for our MitM attack on $T^{\oplus}_{1024}(H')\oplus H'$ (in Fig. 5 (a)) are as follows: $d_r = 32$, $d_b = 32$ and $b_m = 32$ bits. We can get 2^{32} 32-bit partial target preimages with a complexity of 2^{32}. A 32-bit partial target preimage for $T^{\oplus}_{1024}(H')\oplus H'$ can be found with a complexity of 2^0.



(a) Chunk separation of 7-round MitM attack on $T^{\oplus}_{1024}(H')\oplus H'$.

(b) Chunk separation of 7-round MitM attack on $T^{+}_{1024}(M)\oplus M$.

Fig. 5. The MitM Attack on 7-round $T^{\oplus}_{1024}(H')\oplus H'$ and 7-round $T^{+}_{1024}(M)\oplus M$.

MitM Attack on $T^{+}_{1024}(M)\oplus M$ This MitM attack is similar to the MitM attack on $T^{+}_{512}(M)\oplus M$. The AC_2 operation would also cause some problems in this MitM attack. We utilize a similar method in the MitM attack on $T^{+}_{512}(M)\oplus M$ to handle the problems. First, we let the most significant bit of the bytes of $S_6[g_{0,2}, g_{1,1}, g_{2,0}, g_{3,15}, g_{4,14}, g_{5,13}, g_{6,2}, g_{6,3}, g_{6,4}, g_{6,8}, g_{6,9}, g_{6,10}, g_{6,11}, g_{6,12}]$ and $S_8[g_{1,2}, g_{2,1}, g_{3,0}, g_{4,15}, g_{5,14}, g_{6,11}, g_{6,12}, g_{6,13}]$ be 1, the carries of these bytes through AC_2 must be 1. Then we can prevent the forward and backward chunk from affecting each other through AC_2 in the initial structure. The assigned bytes in S_6 and S_8 can be connected by using the freedom of unassigned red bytes

in S_6. Second, the purple bytes at S_0 and S_{11} would be unknown after the AC_2 operation. We should guess the value of the carries for S_0 and S_{11}, which would increase the freedom of the forward and backward chunk. The best parameters for our MitM attack on $T^{+}_{1024}(M)\oplus M$ (see in Fig. 5 (b)) are chosen as follows: $d_r = 31$, $d_b = 29$, $D_{gr} = 1$, $D_{gb} = 3$ and $b_m = 32$ bits in the attack on $T^{+}_{1024}(M)\oplus M$. Then we can get $2^{28}(= 2^{31+29}/2^{32})$ 32-bit partial target preimages with the complexity of 2^{32}. A 32-bit partial target preimage for $T^{+}_{1024}(M) \oplus M$ can be found with a complexity of $2^4(= 2^{32}/2^{28})$.

Minimizing the Overall Complexity To sum up, our pseudo-preimage attack on the compression function for 7-round Kupyna-512 can be obtained with a complexity of $2^{x1} + 2^{x2+4}$, where $x_1 + x_2 \geq 1024-32$. As a result, the overall complexity is the lowest ($\approx 2^{499}$), when we choose $x_1 = 2^{498}$ and $x_2 = 2^{494}$. The memory requirement is about 2^{498}.

5. A SEMI-FREE-START COLLISION ATTACK ON 7-ROUND KUPYNA-512 COMPRESSION FUNCTION

In this section, we present a semi-free-start collision attack on the compression function for 7-round Kupyna-512. Some notations are used to denote the intermediate states of T^{+}_{1024} (or $T^{\oplus}_{1024}$) in our collision attack: r_0 (r'_0) is the initial state of T^{+}_{1024} ($T^{\oplus}_{1024}$), $r_i(r'_i)$ is the state after round i ($1 \leq i \leq r$), and the intermediate states after AC, SB, RB and MC of round i are labeled as $r_i^{AC}(r_i^{AC\prime})$, $r_i^{SB}(r_i^{SB\prime})$, $r_i^{RB}(r_i^{RB\prime})$ and $r_i^{MC} = r_{i+1}(r_i^{MC\prime} = r_{i+1}')$ respectively. We use the same truncated differential trail in $T^{\oplus}_{1024}(H')\oplus H'$ and $T^{+}_{1024}(M)\oplus M$ in our collision attack on Kupyna-512. The truncated differential trail is based on the Super-Sbox technique. Mendel in [7] had pointed out that the truncated differential path for Groestl −512 with 8 active bytes at each end of inbound phase was impossible. As a result, we just adopt a similar truncated differential path used in [7]. The sequence of active bytes between each round i is given as follows:

$$8 \xrightarrow{round\ 1} 1 \xrightarrow{round\ 2} 8 \xrightarrow{round\ 3} 64 \xrightarrow{round\ 4} 112 \xrightarrow{round\ 5} 16 \xrightarrow{round\ 6} 16 \xrightarrow{round\ 7} 112.$$

Based on the truncated differential trail, we can use the rebound attack to find pairs for $T^{\oplus}_{1024}(H')\oplus H'$ and $T^{+}_{1024}(M)\oplus M$ independently. We can find the collision between these pairs by using a birthday attack.

Rebound Attack on $T^{\oplus}_{1024}(H')\oplus H'$ Our rebound attack on $T^{\oplus}_{1024}(H')\oplus H'$ is similar to the rebound attack on Groestl in [7]. The inbound phase is from $r_2^{SB\prime}$ to r_5' (see in Fig. 6), which can be preceded as follows:

Step 1: For all 2^{64} input difference of the $r_2^{SB\prime}$, propagate the difference through $MC \circ RB$ to $r_3^{SB\prime}$. Store the resulting 2^{64} input difference of $r_3^{SB\prime}$ in a list L.

Step 2: Choose a random difference for r_5' and compute the output difference of $r_4^{SB\prime}$.

Step 3: Find the output difference of $r_4^{SB\prime}$ and the input difference of $r_3^{SB\prime}$ that belongs to one Super-Sbox at $r_3^{SB\prime}$. Note that this can be done for each Super-Sbox independently in the following way: (a) Exhaust all 2^{64} value pairs for one Super-Sbox at $r_4^{SB\prime}$, and obtain the valid value pairs of the corresponding Super-Sbox at $r_3^{SB\prime}$. (b) For each value pairs obtained in a), check whether these differences satisfies the possible input difference of

the corresponding Super-Sbox at $r_3^{SB\prime}$ in the list L.

We can create 2^{64} solutions for the inbound phase with a complexity of 2^{64} in time and memory. Then the average time to construct a solution for inbound phase is 1. Starting with 24 active bytes in r_2' and r_6', we can obtain up to 2^{192} solutions in the inbound phase. The complexity of our rebound attack is determined by the outbound phase. In the outbound phase, the state pairs of the inbound phase are propagated outwards probabilistically. The transition 8→1 through the MC operation has a probability of 2^{-56} (from r_2' to r_1'). In order to get one solution that follows the truncated differential trail for $T^{\oplus}_{1024}(H')\oplus H'$, we should create 2^{56} pairs during the inbound phase.

Rebound Attack on $T^{+}_{1024}(M)\oplus M$ Since the AC_2 operation may cause some propagation between bytes, it would influence our rebound attack on $T^{+}_{1024}(M)\oplus M$, we utilize the technique introduced in [4] to solve the problems. The inbound phase is constructed from r_3 to r_5 (see in Fig. 6), it includes two AC_2 operations: one is between r_3 and $r_3^{AC_2}$, the other one is between r_4 and $r_4^{AC_2}$. Since the AC_2 between r_4 and $r_4^{AC_2}$ only affects each Super-Sbox individually, it does not affect this collision attack. However, the AC_2 between r_3 and $r_3^{AC_2}$ may cause difference propagations between 8 Super-Sboxes, and reduce the valid values for inbound phase. In [4], Dobraunig *et al.* let the carries be 1 through the AC_2 operation, and then they can treat each Super-Sbox separately. The valid values for state pairs are reduced since they want some carries to be 1. We summarized the number of valid state pairs for Byte 0-7 in Table 2. Based on Table 2, the valid pairs for the rightmost column Super-Box of r_3^{SB} (in red color) in Fig. 6 is $230.6 \cdot 226.8 \cdot 256 \approx 2^{23.67}$. We can get the valid pairs for the other Super-Box of r_3^{SB} in a similar way. These results are stored in Table 3, from the leftmost to the rightmost column. Then we can create $2^{23.67} \cdot 2^{15.67} \cdot 2^{15.67} \cdot 2^{23.5} \cdot 2^{31.5} \cdot 2^{23.48} \cdot 2^{31.3} \cdot 2^{39.13} \cdot 2^{46.95} \cdot 2^{62.8} \cdot 2^{47.13} \cdot 2^{39.3} \cdot 2^{31.48} \cdot 2^{23.65} \cdot 2^{31.5} - 2^{56*8} \approx 2^{54.4}$ valid solutions for the inbound phase. Since we can obtain these solutions with a complexity of $243 \cdot 241^6 \cdot 256 = 2^{63.4}$, one solution for the inbound phase can be found with a complexity of $2^{63.4}/2^{54.4} = 2^9$.

The success probability of the outbound phase of T^{+} would also be affected by AC_2. We will calculate the success probability as follows. Firstly, we consider the AC_2 between r_5 and $r_5^{AC_2}$, and the AC_2 between r_0 and $r_0^{AC_2}$. Since the active bytes are on the diagonal of r_5 and r_0, these two AC_2 may cause some propagations between bytes. The probabilities of the active bytes in r_5 to $r_5^{AC_2}$ do not affect its neighboring byte are $p_{r5} = (1-2\cdot\frac{13}{256}\cdot\frac{243}{256})^2 \cdot (1-2\cdot\frac{16}{256}\cdot\frac{240}{256})^{12} \cdot 1 \approx 2^{-2.45}$, and the probabilities of the active bytes in r_0 to $r_0^{AC_2}$ do not affect its neighboring byte are $p_{r_0} = (1-2\cdot\frac{13}{256}\cdot\frac{243}{256}) \cdot (1-2\cdot\frac{16}{256}\cdot\frac{240}{256})^6 \cdot 1 \approx 2^{-1.225}$. Secondly, we consider the AC_2 between r_6 and $r_6^{AC_2}$, and the AC_2 between r_2 and $r_2^{AC_2}$. Since the active bytes are within these columns, these two AC_2 operations do not affect the success probability. Furthermore, the probability of the transition from 8→1 through MC is 2^{-56}. Then the probability that a solution of the inbound phase passes the outbound phase is about $2^{-59.675}$. We can obtain $2^{54.4-59.675} = 2^{-5.275}$ pairs, which follows the truncated differential trail with a complexity of $2^{63.4}$. In other words, we can obtain one solution for the rebound attack on $T^{+}_{1024}(M)\oplus M$ with a complexity of $2^{68.675}$.

Minimizing the Overall Complexity In Fig. 6, there are 8 active bytes at the begin-

ning and 16 active bytes at the end of the differential path. If we have collected 2^{y_1} and 2^{y_2} solutions for $T^{+}_{1024}(M)\oplus M$ and $T^{\oplus}_{1024}(H')\oplus H'$ respectively, we should make sure $y_1 + y_2 \geq 192$ so as to obtain a collision by using the birthday attack. Since the complexities for getting a solution for $T^{+}_{1024}(M)\oplus M$ and $T^{\oplus}_{1024}(H')\oplus H'$ are $2^{68.675}$ and 2^{56} respectively, we choose $y_1 = 89.7$ and $y_2 = 102.3$ in our collision attack in order to achieve some balance between the two rebound attacks. Then a semi-free-start collision attack on 7-round Kupyna-512 compression function can be obtained with a complexity of $2^{159.3}$.

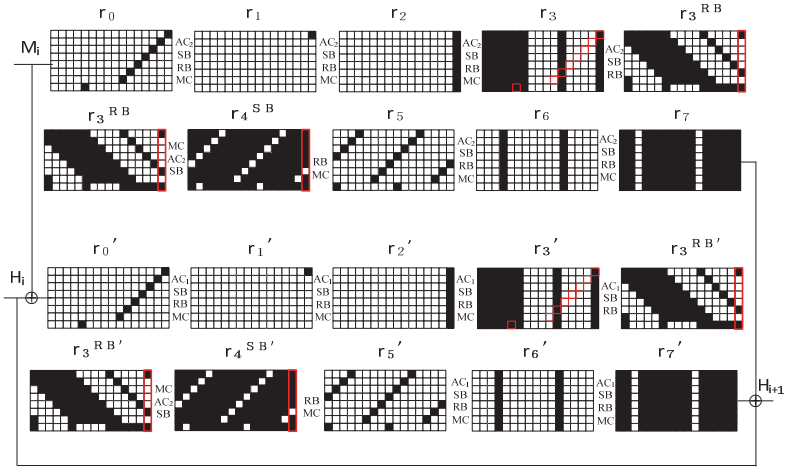


Fig. 6. Collision attack on 7 rounds of the Kupyna-512 compression function.

Table 2. Number of byte values and average number of value pairs.

Byte position	Valid values	Valid pairs (average)
Byte 0	243	230.6
Byte 1-6	241	226.8
Byte 7	256	256

Table 3. The valid pairs for each super-box of r_3^S.

1	2	3	4	5	6	7	8
$2^{23.67}$	$2^{15.67}$	$2^{15.67}$	$2^{15.67}$	$2^{23.5}$	$2^{31.5}$	$2^{23.48}$	$2^{31.3}$
9	10	11	12	13	14	15	16
$2^{39.13}$	$2^{46.95}$	$2^{62.8}$	$2^{47.13}$	$2^{39.3}$	$2^{31.48}$	$2^{23.65}$	$2^{31.5}$

6. COLLISION ATTACK ON 5-ROUND KUPYNA-512

In this section, we will show how to construct a collision attack on the hash function for 5-round Kupyna-512. Our 5-round collision attack is based on the collision attacks on Groestl [8]. In [8], Mendel *et al.* described an alternative description of Groestl in order to simplify the attack process. Denote $T_l^{\oplus\prime}$ and $T_l^{+\prime}$ as the permutation $T_l^{\oplus}$ and T_l^{+} without the last application of *MC*. We also give an alternative description of Kupyna as follows:

$h_0 \leftarrow MC^{-1}(IV)$, $h_{i+1} \leftarrow T_l^{\oplus\prime}(h_i \oplus m_i) \oplus T_l^{+\prime}(m_i) \oplus h_i$ for $i = 0, 1, \ldots, k-1$.

$h = Trunc_n(T_l^{\oplus\prime}(MC(h_k) \oplus MC(h_k))$. Our collision attacks on 5-round Kupyna-512 are based on Super-Sbox technique. We just omit the details of the Super-Sbox. As shown in Fig.7, we should cancel the chosen 16 differences in each iteration, which occurs with a probability of 2^{-128}. In order to get a solution, we should generate 2^{128} pairs following the differential trail, which can be done by using the freedom of the message blocks. We do not need to construct a rebound attack on T^{+}_{1024}, so that AC_2 operation does not affect this collision attack. Our 5-round collision attack (see in Fig. 7) proceeds as follows:

Step 1: Choose m_0, m_0^* randomly, and calculate h_1 until h_1 is fully active.
Step 2: Find (m_1, m_1^*) to cancel the difference of the chosen 16 bytes (white color) in h_2.
Step 3: Find (m_{i+1}, m_{i+1}^*) to cancel the difference of the chosen 16 bytes in h_i $(2 < i \leq 9)$.
Step 4: Repeat the above steps many times until a collision is found in h_9.

Using the Super-Sbox technique, we can find 2^{128} pairs solutions for the inbound phase with a complexity of 2^{128}. All 2^{128} state pairs are propagated in the outbound phase probabilistically. The probability of the outbound phase is $2^{2*-56} = 2^{-112}$, because we should handle two 8→1 transitions. We have $2^{128-112} = 2^{16}$ pairs left for this truncated differential trail. Since we should cancel the difference of the chosen 16 bytes in h_i $(2 \leq i \leq 9)$, we need more message freedom in our collision attack. In [8], Mendel *et al.* solved this problem by using 2^{56} new starting points to cancel the differences in 8 bytes of the state. We want to use the same ideas to solve the freedom problem in our attack. However, we need 2^{112} new starting point, and the length of such colliding message is 2^{115} blocks. The 2^{115} blocks exceeds the maximum length of the message that Kupyna can process. However, the tree-based approach used in [8] can be used to reduce such colliding message pair to 65 message blocks. The complexity of our collision attack is $8 \cdot 2^{128+112} = 2^{243}$, which can be improved to 2^{240} by using denser characteristics. In detail, we do not use the truncated differential trail with two 8→1 transitions in round 3 of the trail in Fig. 7. We can utilize truncated differential trails with 8→8, 8→7, …, 8→2, which satisfies that the probability is greater than 2^{-48}. Since we should still need two 8→1 transitions in the last iteration, the complexity of our collision attack can be improved by a factor of 8. This means we only need 2^{240} compression function calls by using denser characteristics. Besides, the collisions in the chosen-prefix setting can be constructed with the same complexity due to the generic nature of our attack.

7. CONCLUSION

In this paper, we present some pseudo-preimage attacks and collision attacks on Kupyna by combining some known attacks. We construct the pseudo-preimage attacks on the compression function for 5-round Kupyna-256 and 7-round Kupyna-512 by using the guess-and-determine MitM attack. Our pseudo-preimage attacks show that the AC_2 operation provides some additional security against the MitM attack on permutation T_l^{+}. Regarding the collision attack, the rebound attack can not only be used to construct a semi-free-start collision attack on the compression function for 7-round Kupyna-512, but also be used to construct a collision attack on the hash function for 5-round Kupyna-512.

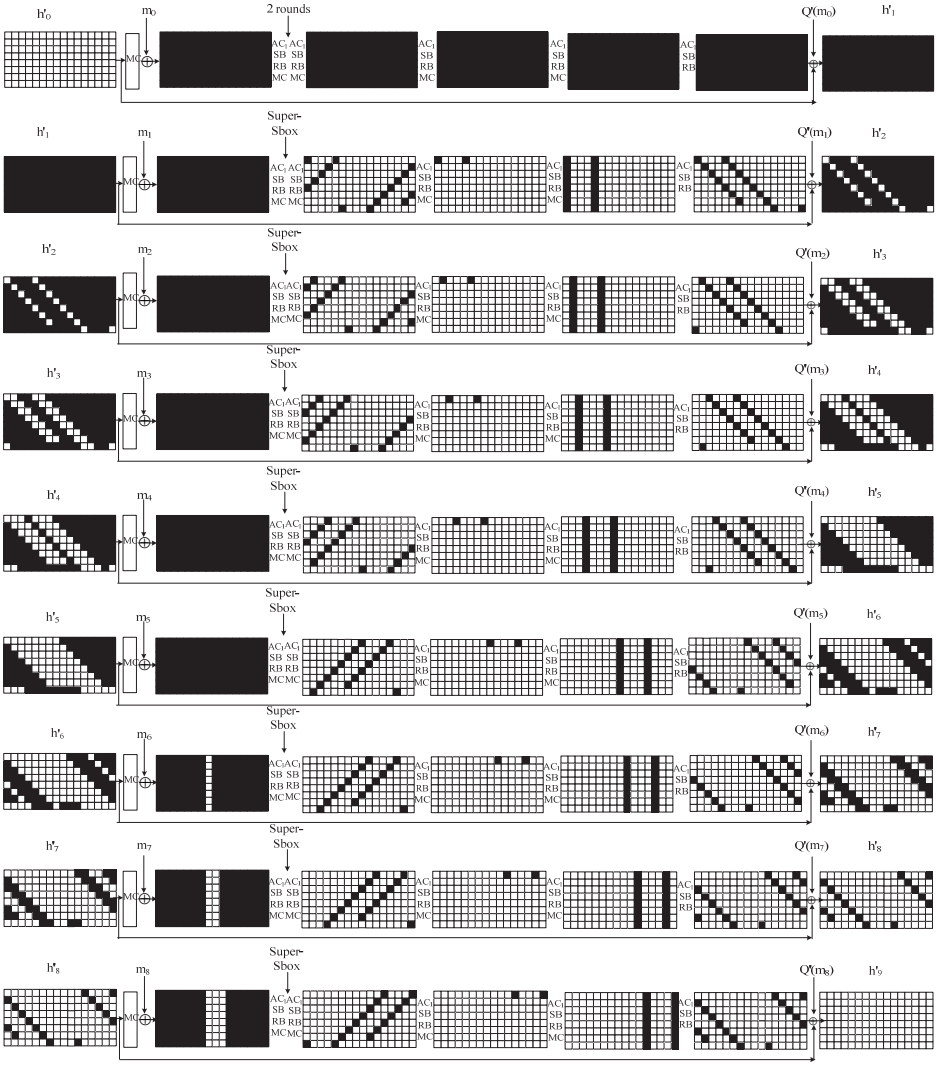
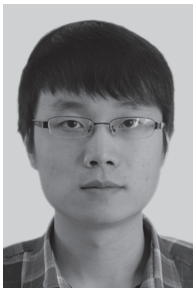


Fig. 7. Collision attack on 5 rounds of the Kupyna-512 hash function.

REFERENCES

1. K. Aoki and Y. Sasaki, "Preimage attacks on one-block MD4, 63-step MD5 and more," in *Proceedings of Selected Areas in Cryptography*, LNCS, Vol. 5381, 2008, pp. 103-119.
2. J. Guo, S. Ling, C. Rechberger, and H. X. Wang, "Advanced meet-in-the-middle preimage attacks: First results on full tiger and improved results on MD4 and SHA-2," in *Proceedings of the 16th International Conference on Theory and Application of Cryptology and Information Security*, LNCS, Vol. 6477, 2010, pp. 56-75.

3. L. Wang, Y. Sasaki, W. Komatsubara, K. Ohta, and K. Sakiyama, "(Second) Preimage attacks on step-reduced RIPEMD/RIPEMD-128 with a new local-collision approach," in *Proceedings of the 11th Cryptographers' Track at the RSA Conference*, LNCS, Vol. 6558, 2011, pp. 197-212.
4. C. Dobraunig, M. Eichlseder, and F. Mendel, "Analysis of the kupyna-256 hash function," in *Proceedings of International Conference on Fast Software Encryption*, LNCS, Vol. 9783, 2016, pp. 575-590.
5. H. Gilbert and T. Peyrin, "Super-sbox cryptanalysis: Improved attacks for aes-like permutations," in *Proceedings of International Conference on Fast Software Encryption*, LNCS, Vol. 6147, 2010, pp. 365-383.
6. F. Mendel, T. Peyrin, C. Rechberger, and M. Schläffer, "Improved cryptanalysis of the reduced groestl compression function, echo permutation and aes block cipher," in *Proceedings of International Workshop on Selected Areas in Cryptography*, LNCS, Vol. 5867, 2009, pp. 16-35.
7. F. Mendel, C. Rechberger, M. Schläffer, and S. S. Thomsen, "Rebound attacks on the reduced groestl hash function," in *Proceedings of the 10th Cryptographers' Track at the RSA Conference*, LNCS, Vol. 5985, 2010, pp. 350-365.
8. F. Mendel, V. Rijmen, and M. Schläffer, "Collision attack on 5 rounds of groestl," in *Proceedings of International Conference on Fast Software Encryption*, LNCS, Vol. 8540, 2014, pp. 509-521.
9. G. I. Kazymyrov, O. Ruzhentsev, V. Kuznetsov, O. Gorbenko, Y. Boiko, A. Dyrda, O. Dolgov, V. Pushkaryov, and A. Oliynykov, "A new standard of ukraine: The kupyna hash function," in *Proceedings of Cryptology ePrint Archive*, Report 2015/885, 2015.
10. S. Wu, D. G. Feng, W. L. Wu, J. Guo, L. Dong, and J. Zou, "(pseudo) preimage attack on round-reduced groestl hash function and others" in *Proceedings of International Conference on Fast Software Encryption*, LNCS, Vol. 7549, 2012, pp. 127-145.
11. J. Zou and L. Dong, "Cryptanalysis of the round-reduced kupyna hash function," in *Proceedings of Cryptology ePrint Archive*, Report 2015/959, 2015.
12. J. Zou, W. L. Wu, S. Wu, and L. Dong, "Improved (pseudo) preimage attack and second preimage attack on round-reduced groestl hash function," *Journal of Information Science and Engineering*, Vol. 30, 2014, pp. 1789-1806.
13. D. Wagner, "A generalized birthday problem," in *Proceedings of CRYPTO*, LNCS, Vol. 2442, 2002, pp. 288-303.



Jian Zou (鄒劍) is now a Lecturer at Fuzhou University. He received his Ph.D. degree from Graduate School of Chinese Academy of Sciences and Institute of Software, Chinese Academy of Sciences in 2015. His current research areas include hash function and block ciphers.

Le Dong (董樂) is an Associate Professor at Henan Normal University. He received his Ph.D. degree from Graduate School of Chinese Academy of Sciences and Institute of Software, Chinese Academy of Sciences in 2013. His current research areas include hash function and block ciphers.