Optimizing Rectangle Attacks: A Unified and Generic Framework for Key Recovery

Ling Song[1], Nana Zhang[2,3], Qianqian Yang[2,3]✉, Danping Shi[2,3], Jiahao Zhao[2,3], Lei Hu[2,3], and Jian Weng[1]

[1] College of Information Science and Technology, Jinan University, Guangzhou, China
[2] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[3] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
songling.qs@gmail.com,zhangnana@iie.ac.cn,yangqianqian@iie.ac.cn, shidanping@iie.ac.cn, hulei@iie.ac.cn, cryptjweng@gmail.com

Abstract. The rectangle attack has shown to be a very powerful form of cryptanalysis against block ciphers. Given a rectangle distinguisher, one expects to mount key recovery attacks as efficiently as possible. In the literature, there have been four algorithms for rectangle key recovery attacks. However, their performance vary from case to case. Besides, numerous are the applications where the attacks lack optimality. In this paper, we investigate the rectangle key recovery in depth and propose a unified and generic key recovery algorithm, which supports any possible attacking parameters. Notably, it not only covers the four previous rectangle key recovery algorithms, but also unveils five types of new attacks which were missed previously. Along with the new key recovery algorithm, we propose a framework for automatically finding the best attacking parameters, with which the time complexity of the rectangle attack will be minimized using the new algorithm. To demonstrate the efficiency of the new key recovery algorithm, we apply it to `Serpent`, `CRAFT`, `SKINNY` and `Deoxys-BC`-256 based on existing distinguishers and obtain a series of improved rectangle attacks.

Keywords: Boomerang attack, Rectangle attack, Key recovery algorithm, `Serpent`, `CRAFT`, `SKINNY`, `Deoxys-BC`

1 Introduction

Differential cryptanalysis, which was introduced by Biham and Shamir [BS91], is one of the most powerful cryptanalytic approaches for assessing the security of block ciphers. The basic idea is to exploit non-random propagation of input difference to output difference, *i.e.*, high-probability differentials. In many cases, it is may be hard to find a long differential of high probability. In 1999, Wagner proposed the boomerang attack [Wag99], which divides a cipher E into two sub-ciphers and utilizes two short differentials of high probability to construct a long one.

Suppose $E = E_1 \circ E_0$, where there are two short differentials $\alpha \to \beta$ and $\gamma \to \delta$ with probability p and q for E_0 and E_1, respectively. The boomerang

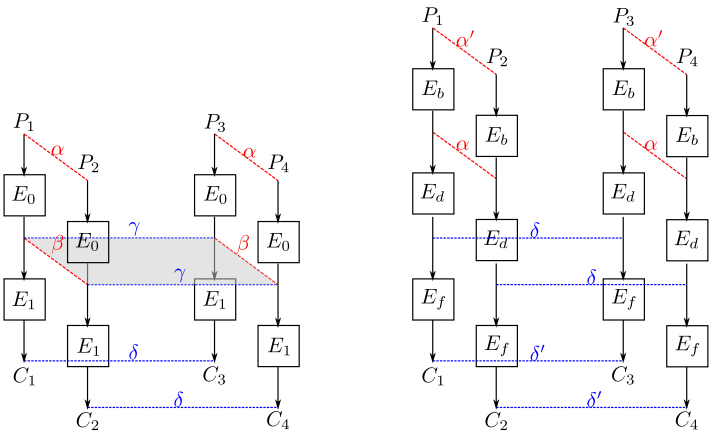

Figure 1: Basic boomerang attack (left) and the schematic view of the key recovery (right)

attack, as depicted in Figure 1 (left), exploits the high probability of the following differential property:

$$\Pr\left[E^{-1}\big(E(x) \oplus \delta\big) \oplus E^{-1}\big(E(x \oplus \alpha) \oplus \delta\big) = \alpha\right] = p^2q^2. \tag{1}$$

The basic boomerang attack requires adaptive chosen plaintexts and ciphertexts. Later, Kelsey *et al.* developed a chosen-plaintext variant, named the amplified boomerang attack [KKS00]. However, this transition reduced the probability of the distinguisher to $2^{-n}p^2q^2$. In [BDK01], Biham *et al.* further converted the amplified boomerang attack into the rectangle attack by considering as many differences as possible in the middle to estimate the probability more accurately. As a result, the probability of a rectangle distinguisher becomes $2^{-n}\hat{p}^2\hat{q}^2$, where $\hat{p} = \sqrt{\Sigma_i \Pr^2(\alpha \to \beta_i)}$ and $\hat{q} = \sqrt{\Sigma_j \Pr^2(\gamma_j \to \delta)}$. The boomerang and rectangle attack then have been applied to numerous block ciphers, such as `Serpent` [BDK01], `AES` [BK09], `KASUMI` [DKS10b, DKS14], etc.

Since the boomerang attack was proposed, there has been a line of research on estimating the probability of boomerang distinguishers more accurately so as to find better distinguishers. At first, the probability of a boomerang distinguisher was considered as p^2q^2 by simply assuming the two differentials are independent until the dependency issue between the two differentials came into view. In boomerang or rectangle attacks on concrete ciphers, observations were made that the probability computed via p^2q^2 may be inaccurate in some cases from [BK09, Mur11], where the probability can be higher by using tricks or the two chosen differentials may be even incompatible. Taking the dependency between the two differentials into account, Dunkelman *et al.* suggested the sandwich attack [DKS10b, DKS14] which estimates the probability by p^2q^2r, where r is the exact probability for a middle part. Later, a new tool named boomerang connectivity table (BCT) was proposed to estimate the probability r theoretically [CHP+18, SQH19].

Another line of research on the boomerang and rectangle attack is to mount key recovery attacks as efficiently as possible. Figure 1 (right) displays a schematic view of key recovery attacks based on a distinguisher over the middle part E_d. The first rectangle key recovery algorithm was proposed by Biham *et al.* in [BDK01] along with the proposal of the rectangle attack. This algorithm was applied to 10-round `Serpent` [ABK98] with an 8-round rectangle distinguisher. Shortly after that, in [BDK02] the same authors introduced the second rectangle key recovery algorithm which can improve the result on `Serpent` by reducing the time complexity. There was no improvement until Zhao *et al.* proposed a new rectangle key recovery algorithm in [ZDM+20] which originally works for ciphers with a linear key schedule in the related-key setting, but it can be converted to the single-key setting trivially. Such an algorithm, when applied to `SKINNY` [BJK+16a] outperforms the two previous key recovery algorithms. However, the algorithm presented in a very recent work [DQSW22] makes a step further on improving rectangle attacks on `SKINNY` and some other ciphers.

Motivation. Even though the two recent rectangle key recovery algorithms provide surprisingly good results on `SKINNY`, we carefully check that they do not beat the algorithm in [BDK02] when applied to `Serpent`. On the other hand, the algorithm in [BDK02] is not efficient on `SKINNY` when compared with the two recent ones. Then, the following questions arise.

– Given a rectangle distinguisher of a block cipher, how efficient the key recovery can be?
– Are there any other ways to mount key recovery attacks?

Not only would answer to these questions be of great significance to cryptanalysis of block ciphers, but also provide a deeper understanding of the key recovery of the rectangle attack.

Our contributions. In this paper, we investigate the rectangle key recovery in depth and completely answer the above questions. The starting point of our work is some new insights that the key recovery of the rectangle attack always includes steps of constructing pairs from single messages and quartets from pairs, whereas the number of pairs or quartets that will be constructed is affected by guessed subkey bits. With this in mind, we propose a unified and generic rectangle key recovery algorithm which supports any possible attacking parameters, particularly any number of guessed subkey bits. Our contributions on the key recovery algorithm are summarized as follow.

– Based on a deeper understanding of the rectangle key recovery, a unified and generic key recovery algorithm is proposed. It supports any number of guessed key bits and covers the four previous rectangle key recovery algorithms, *i.e.*, any of the previous four algorithms is a special case of our algorithm. What's

more, it unveils five types of new attacks which were missed previously (see Figure 8 in Section 5 for more information).
- Although our new algorithm supports any set of attacking parameters, it does not tell which is the best on its own. As a complement, we propose a framework for automatically finding the best parameters for the new algorithm. When we feed the parameters returned by this framework to our new key recovery algorithm, the time complexity of the rectangle attack will be minimized.
- We also develop variants of the new key recovery algorithm for related attacks, including the rectangle attack in the related-key setting for ciphers with a linear key schedule and boomerang attacks in both single-key and related-key setting, etc.

Previously, the four mentioned key recovery algorithms are treated as separate ones. Given a rectangle distinguisher, one can compute the complexities for all algorithms and pick the algorithm with the lowest complexity. Now, we can work with the new algorithm only. To demonstrate the efficiency of the new key recovery algorithm, we apply it to four block ciphers using existing distinguishers and obtain a series of improved results.

- We revisit the attack on 10-round `Serpent` and find better attacks than the one given in [BDK02].
- We revisit the rectangle attacks on round-reduced `SKINNY` in [DQSW22], which are the best existing attacks on `SKINNY` in the related-tweakey setting. For the four distinguishers of `SKINNY`, we find better attacks for three of them, despite the fact that these distinguishers were searched dedicated for the key recovery algorithm in [DQSW22].
- We extend the rectangle attack on `CRAFT` by one round and give the first 19-round attack, which is the best attack on this cipher so far in the single-key setting.
- On `Deoxys-BC`-256, we improved the 11-round rectangle attack and extend the boomerang attack by one round in the related-tweakey setting. These are the best attacks on `Deoxys-BC`-256 so far in terms of time complexity.

These results are summarized in Table 1. According to these applications, we find that the best attacking parameters differ significantly from those which were used in previous works and even the number rounds added around the distintuisher is different. Notably, these new attacking parameters are not covered by the previous key recovery algorithms in many cases. Thus, it is likely that previous rectangle attacks can be improved to some extent using the new key recovery algorithm.

Organization. The rest of the paper is organized as follows. In Section 2, we give notations which will be used throughout the paper. In Section 3, the new rectangle key recovery algorithm will be introduced as well as the framework for automatically finding the best attacking parameters and extensions of the new algorithm. Section 4 presents applications of the new algorithm to four block ciphers. In Section 5, we compare our new rectangle key recovery algorithm with the four previous ones in detail. We conclude this paper in Section 6.

Table 1: Summary of the cryptanalytic results.

Cipher	Rounds	Data	Memory	Time	Approach	Setting	Ref.
Serpent	10	$2^{126.8}$	2^{192}	2^{217}	Rectangle	SK	[BDK01]
		$2^{126.3}$	$2^{126.3}$	$2^{173.8}$	Rectangle	SK	[BDK02]
		$2^{126.3}$	$2^{126.3}$	$2^{159.11}$	Rectangle	SK	Sect. 4.1
		$2^{124.15}$	$2^{124.15}$	$2^{155.67}$	Rectangle	SK	Sect. 4.1
CRAFT	18	$2^{60.92}$	2^{84}	$2^{101.7}$	Rectangle	SK	[HBS21]
	19	$2^{60.92}$	2^{72}	$2^{112.61}$	Rectangle	SK	Sect. 4.2
SKINNY-64-128	25	$2^{61.67}$	$2^{64.26}$	$2^{118.43}$	Rectangle	RK	[DQSW22]
		$2^{61.67}$	$2^{63.67}$	$2^{110.03}$	Rectangle	RK	Sect. 4.3
SKINNY-128-384	32	$2^{123.54}$	$2^{123.54}$	$2^{354.99}$	Rectangle	RK	[HBS21]
		$2^{123.54}$	$2^{129.54}$	$2^{344.78}$	Rectangle	RK	Sect. B.1
SKINNY-128-256	26	$2^{126.53}$	2^{136}	$2^{254.4}$	Rectangle	RK	[HBS21]
		$2^{126.53}$	2^{136}	$2^{241.38}$	Rectangle	RK	Sect. B.1
Deoxys-BC-256	10	$2^{127.58}$	$2^{127.58}$	2^{204}	Rectangle	RK	[CHP+17]
	11	$2^{122.1}$	$2^{128.2}$	$2^{249.9}$	Rectangle	RK	[ZDJ19a]
	11	$2^{126.78}$	2^{128}	$2^{222.49}$	Rectangle	RK	Sect. B.2
	10	$2^{98.4}$	2^{88}	$2^{249.9}$	Boomerang	RK	[ZDJ19a]
	11	$2^{122.4}$	2^{128}	$2^{218.65}$	Boomerang	RK	Sect. 4.4

2 Notations

In this paper, we focus on the key recovery for a given boomerang distinguisher. For simplicity, we treat a target cipher $E : \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n$ as $E = E_f \circ E_d \circ E_b$, where there is a boomerang distinguisher over E_d of probability P^2, *i.e.*,

$$\Pr\left[E_d^{-1}(E_d(P_1) \oplus \delta) \oplus E_d^{-1}(E_d(P_1 \oplus \alpha) \oplus \delta) = \alpha\right] = P^2. \tag{2}$$

That is, we take the probability of the boomerang distinguisher for P^2 and do not pay attention to whether it is evaluated with p^2q^2r or $\hat{q}^2\hat{q}^2$. Figure 1 (right) depicts the framework of E, where E_b and E_f are added around E_d. The aim of the key recovery is to identify partial subkeys used in E_b and E_f by utilizing the distinguisher over E_d and further to find the master key more efficiently than the exhaustive search.

To describe the key recovery, a series of notations are used through out the paper. For convenience, we borrow some notations which are frequently used in the previous works on rectangle attacks, such as [BDK02,LGS17,ZDM+20,DQSW22]. As shown in Figure 2, the input difference of the distuinghser α propagates back over E_b^{-1} to α'. Let V_b be the space spanned by all possible α' where $r_b = \log_2|V_b|$. The output difference of the distinguisher δ propagates forward over E_f to δ'. Let V_f be the space spanned by all possible δ' where $r_f = \log_2|V_f|$. Let k_b be the subset of subkey bits which are employed in E_b and affect the propagation

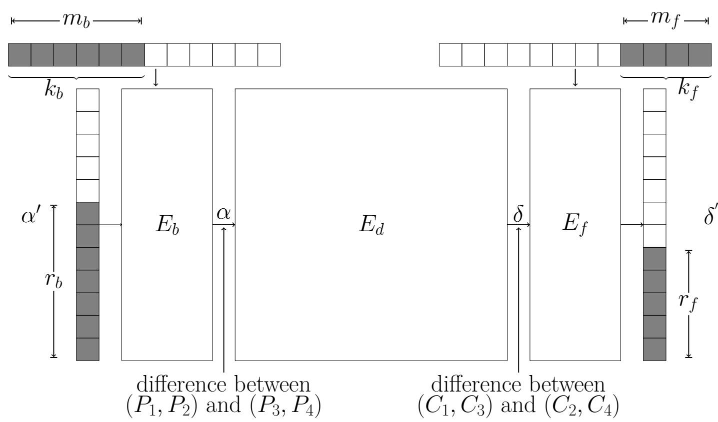

Figure 2: Outline of rectangle key recovery attack

$\alpha' \to \alpha$. Similarly, let k_f be the subset of subkey bits which are used in E_b and affect the propagation $\delta \leftarrow \delta'$. Then let $m_b = |k_b|$ and $m_f = |k_f|$ be the number of bits in k_b and k_f, respectively.

In a specific key recovery algorithm, a part of k_b and k_f, denoted by k'_b, k'_f, may be guessed at first. Let $m'_b = |k'_b|$ and $m'_f = |k'_f|$. With the guessed subkey bits, the differential propagations $\alpha' \to \alpha$ and $\delta \leftarrow \delta'$ can be partially verified. Suppose under the guessed subkey bits r'_b-bit condition on the top and r'_f-bit condition on the bottom can be verified. Finally, let $r^*_b = r_b - r'_b$ and $r^*_f = r_f - r'_f$.

In this paper, we mainly focus on the rectangle key recovery algorithms in the single-key setting and these can be easily converted into the related-key setting for ciphers with linear key schedule.

3 A Unified and Generic Key Recovery Algorithm

In this section, we present our unified and generic key recovery algorithm for the rectangle attack. Before specifying our algorithm, we recall basics of the rectangle attack and provide new insights into the key recovery, which will be the base of our algorithm. Our algorithm is generic and supports any possible key guessing strategy. However, given a specific rectangle distinguisher, which parameters are the best for our algorithm? A framework for automatically finding the best parameters is then introduced afterwards. Finally, we discuss extensions of our algorithm to related cases.

3.1 Basic Ideas and Intuitions

In this subsection, we recall the principles of the rectangle attack and give some new insights on the key recovery which are core ideas behind our new algorithm.

As can be seen from Figure 1 and Eq. (2), the boomerang distinguisher is built on a nonrandom property of quartets. The rectangle distinguisher is its chosen-plaintext variant. This nonrandom property is then used to extract subkey information in E_b and E_f. As in standard differential cryptanalysis, candidates for subkey k_b and k_f are identified if they are suggested by a sufficiently large number of quartets. Here, k_b and k_f are suggested by a quartet $(P_i, C_i), i = 1, 2, 3, 4$, if

$$E_b(k_b, P_1) \oplus E_b(k_b, P_2) = E_b(k_b, P_3) \oplus E_b(k_b, P_4) = \alpha,$$
$$E_f^{-1}(k_f, C_1) \oplus E_f^{-1}(k_f, C_3) = E_f^{-1}(k_f, C_2) \oplus E_f^{-1}(k_f, C_4) = \delta$$

holds. As shown in Figure 2, the α difference propagates to α' via E_b^{-1} and $\alpha' \in V_b$. It does not mean every element of V_b is a possible α', whereas any difference outside V_b is impossible for α. The same applies for the bottom side. This means, quartets with plaintext difference outside V_b or ciphertext difference outside V_f will not suggest any subkeys. Therefore, an important step in rectangle key recovery algorithms is to construct quartets which are possible to suggest subkeys and at least satisfy $P_1 \oplus P_2, P_3 \oplus P_4 \in V_b$ and $C_1 \oplus C_3, C_2 \oplus C_4 \in V_f$.

Data complexity. A commonly-used idea to reduce the data complexity is to employ plaintext structures which allow to enjoy the birthday effect. A plaintext structure takes all possible values for the r_b bits and chooses a constant for the remaining $n - r_b$ bits. For each structure, there are 2^{2r_b-1} pairs of plaintext with difference in V_b and 2^{r_b-1} of them satisfy α difference by meeting a r_b-bit condition.

Given a boomerang distinguisher with probability P^2, the number of quartets satisfying the input difference α of the distinguisher should be at least $sP^{-2}2^n$ for a rectangle attack, where s is the expected number of right quartets (say $s = 4$). These quartets can be formed from plaintext pairs taken in structures. Suppose the number of structures needed is y. Note y structures can constitute $2 \cdot \binom{y2^{r_b-1}}{2}$[4] quartets that satisfy α difference. Then $y = \sqrt{s}2^{n/2-r_b+1}/P$ and the data complexity is $D = y \cdot 2^{r_b} = \sqrt{s}2^{n/2+1}/P$. This infers that the data complexity is the same in different key recovery algorithms.

Time complexity. Next, let us investigate the time complexity from a high-level perspective. We stress that the key recovery of the rectangle attack always includes steps of constructing pairs from single messages and quartets from pairs. Therefore, the whole key recovery can be split into the following phases: (1) data collection, (2) pair construction, (3) constructing quartets and processing them to extract subkeys, and last (4) a brute force search for the unique right master key among key candidates. The time complexities of the first and the last phases are easy to estimate, so let us focus on the time complexities of the middle two phases, which we denote by T_2 and T_3, respectively.

T_3 is mainly affected by the number of quartet candidates. From D plaintexts, we can construct $N = D^2 \cdot 2^{2r_b+2r_f-2n-2}$ quartet candidates with plaintext

[4] If both (P_1, P_2) and (P_3, P_4) satisfy α difference, then we can form two quartets: (P_1, P_2, P_3, P_4) and (P_1, P_2, P_4, P_3).

difference in V_b and ciphertext difference in V_f. This seems to be a fixed term like the data complexity. However, the number of quartets to be processed may be reduced when some subkey bits are guessed. Recall that m_b-bit k_b and m_f-bit k_f are involved for the propagation $\alpha' \leftarrow \alpha$ and $\delta \rightarrow \delta'$ and verifying α difference and δ difference for such a quartet takes $2r_b$-bit and $2r_f$-bit conditions (as there are two pairs), respectively. Thus, there will be $N \cdot 2^{m_b+m_f-2r_b-2r_f} = D^2 \cdot 2^{m_b+m_f-2n-2}$ suggestions for k_b and k_f in total. On average, the number of suggestions for a wrong subkey is less than 1 as $D^2 \cdot 2^{-2n-2} < 1$, while it is s for the right subkey. On the one hand, this confirms that the rectangle attack works; on the other hand, it means when the subkey is fixed, most quartets are wrong and thus may likely be filtered out before being constructed. This is what has been done in the first rectangle key recovery algorithm proposed in [BDK01] and rewritten in Appendix C.1, which guesses the whole k_b and k_f.

However, a full guess of k_b and k_f is not necessary to reduce the number of quartet candidates, as studied in [ZDM+20,DQSW22]. In this paper, we consider the most general situation where a part of k_b, *i.e.*, k'_b, and a part of k_f, *i.e.*, k'_f are guessed, with $m'_b = |k'_b|$, $m'_f = |k'_f|$, $0 \leq m'_b \leq m_b$ and $0 \leq m'_f \leq m_f$. To have a better view of this situation, we present a toy example in Figure 3 to illustrate the parameters. Assume under the guess r'_b-bit (resp. r'_f-bit) condition can be verified for a plaintext (resp. ciphertext) pair. Then the number of quartets to be processed is $2^{m'_b+m'_f} \cdot D^2 \cdot 2^{2r^*_b+2r^*_f-2n-2}$, where $r^*_b = r_b - r'_b$ and $r^*_f = r_f - r'_f$. *We point out the number of quartet candidates gets smaller as long as* $m'_b + m'_f < 2r'_b + 2r'_f$.

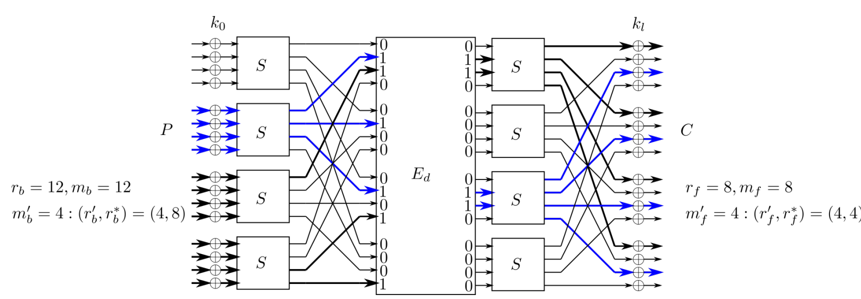

Figure 3: A toy example to illustrate the parameters of the rectangle key recovery where subkey bits corresponding to blue lines are guessed.

Let us come to the time complexity of constructing pairs, *i.e.*, T_2. Note that T_2 is determined by the number of pairs that are used to construct quartets. we emphasize that pairs can be constructed either on the top for plaintexts or on the bottom for ciphertexts. Still assume partial subkey bits are guessed. Then the number of filters for plaintext pairs is $n - r^*_b$ while it is roughly $n - r^*_f$ for ciphertext pairs (we will present the exact number of filters in the next subsection). Since filters for plaintext pairs and filters for ciphertext pairs work on different faces, they can not be taken into account simultaneously in the phase of constructing pairs. *The key principle is to form pairs on the side with more filters so that T_2 is lower.*

Questions. Then, there come two questions:

Question 1: *How does the key recovery algorithm proceed when k'_b and k'_f are guessed, where $m'_b = |k'_b|$, $m'_f = |k'_f|$, $0 \le m'_b \le m_b$ and $0 \le m'_f \le m_f$?*

Question 2: *What is the best choice for (k'_b, k'_f) so that the overall time complexity is minimized?*

To answer the first question, we propose a detailed algorithm for the rectangle key recovery in the next subsection. Because this algorithm supports any possible (k'_b, k'_f) and covers all previous key recovery algorithms, we call it a generic and unified algorithm for the rectangle key recovery. For the second question, we present a framework for automatically finding the best (k'_b, k'_f) in Section 3.3. Combining both, we are able to find the most efficient rectangle key recovery attack.

3.2 Generic and Unified Algorithm for the Rectangle Key Recovery Attack

In the following, we describe our algorithm for the rectangle key recovery attack which works for any number of guessed key bits. Suppose m'_b-bit k'_b and m'_f-bit k'_f are to be guessed. Then the specific steps of our algorithm are as follows. Note the toy example in Figure 3 would be helpful for understanding the algorithm.

1. Collect and store y structures of 2^{r_b} plaintexts. Hence, the data complexity is $D = y \cdot 2^{r_b}$. The time and memory complexities of this step are also D.
2. Split $(m'_b + m'_f)$-bit $k'_b \| k'_f$ into two parts: $G_L \| G_R$ where G_L has t bits.
3. Guess G_R:
   (a) Initialized a list of key counters for G_L and unguessed key bits of k_b, k_f. The memory complexity in this step is $2^{t+m_b+m_f-m'_b-m'_f}$.
   (b) Guess the t-bit G_L:
      i. For each data (P_1, C_1), partially encrypt P_1 and partially decrypt C_1 under the guessed subkey bits. Let $P_1^* = Enc_{k'_b}(P_1)$ and $C_1^* = Dec_{k'_f}(C_1)$. For each structure, we will get $2^{r'_b}$ sub-structures, each of which includes $2^{r_b - r'_b} = 2^{r^*_b}$ plaintexts which take all possible values for the active bits. In other words, there are $y^* = y \cdot 2^{r'_b}$ structures of $2^{r^*_b}$ plaintexts. The time complexity of this step is D.
      ii. Let $2^{-\mu} = D \cdot 2^{-n}$. If $r^*_f - \mu \ge r^*_b$, it turns to step (A); else if $r^*_f - \mu < r^*_b$, it turns to step (D).
         A. Insert all the obtained (P_1^*, C_1^*) into a hash table according to $n - r^*_b$ bits of P_1^*. Then construct a set as $S = \{(P_1^*, C_1^*, P_2^*, C_2^*) :$ P_1^* and P_2^* have difference only in r^*_b bits$\}$. The size of S is $y \cdot 2^{r'_b} \cdot 2^{2(r_b - r'_b)-1} = D \cdot 2^{r^*_b - 1}$. Hence, the time and memory complexities of this step are both $D \cdot 2^{r^*_b - 1}$.

B. Insert S into a hash table by $n-(r_f-r'_f)=n-r^*_f$ inactive bits of C^*_1 and $n-(r_f-r'_f)=n-r^*_f$ inactive bits of C^*_2.
C. For each $2(n-r^*_f)$-bit index, we pick two distinct $(P^*_1, C^*_1, P^*_2, C^*_2)$, $(P^*_3, C^*_3, P^*_4, C^*_4)$ to generate the quartet. We will get

$$2 \cdot \binom{\frac{|S|}{2^{2(n-r^*_f)}}}{2} \cdot 2^{2(n-r^*_f)} = D^2 \cdot 2^{2r^*_b} \cdot 2^{2r^*_f} \cdot 2^{-2n-2}$$

quartets. Then go to step (iii).
D. Insert all the obtained (P^*_1, C^*_1) into a hash table according to $n-r^*_f$ bits of C^*_1. Then construct a set as $S=\{(P^*_1, C^*_1, P^*_3, C^*_3):$ C^*_1 and C^*_3 are colliding in $n-r^*_f$ bits$\}$. The size of S is $D^2 \cdot 2^{r_f-r'_f-n-1} = D \cdot 2^{r^*_f-1-\mu}$. Hence, the time and memory complexities of this step are both $D \cdot 2^{r^*_f-1-\mu}$.
E. Insert S into a hash table by $n-r^*_b$ inactive bits of P^*_1 and $n-r^*_b$ inactive bits of P^*_3.
F. There are at most $2^{2(n-r^*_b-\mu)}$ possible values for the $2(n-r^*_b)$-bit index. For each index, we pick two distinct entries $(P^*_1, C^*_1, P^*_3, C^*_3)$, $(P^*_2, C^*_2, P^*_4, C^*_4)$ to generate the quartet. We will get

$$2 \cdot \binom{\frac{|S|}{2^{2(n-r^*_b-\mu)}}}{2} \cdot 2^{2(n-r^*_b-\mu)} = D^2 \cdot 2^{2r^*_b} \cdot 2^{2r^*_f} \cdot 2^{-2n-2}$$

quartets.

iii. Determine the key candidates involved in E_b and E_f and increase the corresponding counters. Denote the time complexity for processing one quartet as ϵ. Then the time complexity in this step is $D^2 \cdot 2^{2r^*_b} \cdot 2^{2r^*_f} \cdot 2^{-2n-2} \cdot \epsilon$.

(c) Select the top $2^{t+m_b+m_f-m'_b-m'_f-h}$ hits in the counters to be the candidates, which delivers a h-bit or higher advantage.
(d) Guess the remaining $k-m_b-m_f$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key. The time complexity of this step is $2^{k-m'_b-m'_f-h}$.

Data complexity. The data complexity is $D = y \cdot 2^{r_b} = \sqrt{s}2^{n/2+1}/P$.

Memory complexity. The memory complexity is $M = D + min\{D \cdot 2^{r^*_b-1}, D \cdot 2^{r^*_f-1-\mu}\} + 2^{t+m_b+m_f-m'_b-m'_f}$ for storing the data, the set S, and the key counters.

Time complexity. The time complexity of collecting data is $T_0 = D$, the time complexity of doing partial encryption and decryption under guessed key bits is

$$T_1 = 2^{m'_b+m'_f} \cdot D = 2^{m'_b+m'_f} \cdot y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{m'_b+m'_f+\frac{n}{2}+1}/P,$$

the time complexity of generating set S is

$$T_2 = 2^{m'_b+m'_f} \cdot D \cdot min\{2^{r^*_b-1}, 2^{r^*_f-1-\mu}\}$$
$$= min\{\sqrt{s} \cdot 2^{m'_b+m'_f+r_b-r'_b+\frac{n}{2}}/P, s \cdot 2^{m'_b+m'_f+r_f-r'_f+1}/P^2\},$$

the time complexity of generating and processing quartet candidates is

$$T_3 = 2^{m'_b+m'_f} \cdot D^2 \cdot 2^{2r^*_b} \cdot 2^{2r^*_f} \cdot 2^{-2n-2} \cdot \epsilon = (s \cdot 2^{m'_b+m'_f-n+2r_b+2r_f-2r'_b-2r'_f+1}/P^2) \cdot \epsilon,$$

and the time complexity of exhaustive search is $T_4 = 2^{m'_b+m'_f} \cdot 2^{k-m'_b-m'_f-h} = 2^{k-h}$, where $h \le 2^{t+m_b+m_f-m'_b-m'_f}$. The overall time complexity is the sum of $T_i, i \in [0,4]$.

On t and h. According to [Sel08], the success probability of differential analysis is

$$P_s = \int_{\frac{\sqrt{sS_N}-\Phi^{-1}(1-2^{-h})}{\sqrt{S_N+1}}}^{\infty} \phi(x)dx,$$

where S_N is the signal-to-noise ratio and $S_N = \frac{2^{-n}P^2}{2^{-2n}}$ in rectangle attacks as well as in boomerang attacks. In the algorithm, t is an integer ranging from 0 to $m'_b + m'_f$. It not only gives much greater flexibility in choosing h, but also allows the previous rectangle key recovery algorithm to fit in easily regarding setting the key counters. We will discuss more about the relation with the previous algorithms in Section 5.

On ϵ. In the algorithm, m'_b bits of k_b and m'_f bits of k_f are guessed, respectively. With the guessed subkey bits, partial differential propagation over E_b (resp. E_f) can be ensured by properly selecting pairs. Now suppose input difference (resp. output differencce) fall in a smaller space V^*_b (resp. V^*_f) where $r^*_b = |V^*_b|$ (resp. $r^*_f = |V^*_f|$). In step 2(d) of the algorithm, the subkey information is extracted from quartets with input difference in V^*_b and output difference in V^*_f. Then, ϵ is defined to be the time to process one such quartet.

Recall that a right quartet satisfies $E_b(P_1) \oplus E_b(P_2) = \alpha = E_b(P_3) \oplus E_b(P_4)$. Both pairs are encrypted by the same subkey, so a right quartet must agree on the remaining m^*_b bits of k_b. Under the guess of m'_b bits of k_b, there are $2^{r^*_b}$ possible input differences that lead to α difference after E_b. Since each pair suggests $2^{m^*_b-r^*_b}$ subkeys on average, both pairs agree on $2^{2(m^*_b-r^*_b)}/2^{m^*_b} = 2^{m^*_b-2r^*_b}$ for E_b. Similarly, for E_f we get $2^{m^*_f-2r^*_f}$ suggestions for the remaining m^*_f bits of k_f. Consequently, each quartet suggests $2^{m^*_b+m^*_f-2r^*_b-2r^*_f}$ possible subkeys.

There are different methods to deduce the remaining m^*_b bits of k_b suggested by these quartets. A recommended method is to precompute a hash table for all possible input pairs and the value of m^*_b-bit k_b that can lead to α difference. This table can be built with time complexity $2^{r^*_b+m^*_b}$ and indexed by the values of the pairs. The memory cost of this table is $2^{r^*_b+m^*_b}$ (rather than $2^{r^*_b}$ in [BDK01]). When processing a quartet, we can extract the subkey candidates suggested by both pairs by looking up the table twice. Do the same thing for E_f. Therefore,

ϵ will be no more than $\max\{4, 2^{m_b^*-r_b^*} + 2^{m_f^*-r_f^*}\}$ memory accesses, provided that two lookup tables have been built with time and memory complexity of $2^{r_b^*+m_b^*} + 2^{r_f^*+m_f^*}$. If $2^{m_b^*-r_b^*} + 2^{m_f^*-r_f^*}$ is relatively large, ϵ can be lowered to no more than $\max\{2, 2^{m_b^*-2r_b^*} + 2^{m_f^*-2r_f^*}\}$ by using tables built for quartets. In this case, the memory cost increases to $2^{2r_b^*+m_b^*} + 2^{2r_f^*+m_f^*}$, which also means achieving the smallest ϵ at the cost of memory. This is specially profitable when $2^{2r_b^*+m_b^*} + 2^{2r_f^*+m_f^*}$ is not dominant for memory cost.

Note that sometimes the above method of processing quartets may not be applied directly. In certain cases, besides the r_b^* bits, some other non-active bits of pairs are needed to verify α difference after E_b, resulting in a larger time complexity for building a precomputation table as well as a larger memory cost. For the bottom part E_f, it is similar. As an example, this can be seen from rectangle attacks on `SKINNY` (*e.g.*, Figure 6). In such cases, we suggest building lookup tables for smaller local operations. Consequently, ϵ can be equivalent to a few memory accesses.

Minimizing the time complexity. As can be seen from the formulas of $T_i, i \in [0, 4]$, the overall time complexity depends on the number of guessed subkey bits $m'_b + m'_f$ and the number of filters $r'_b + r'_f$ obtained under these guessed subkey bits. In order to reduce the time complexity, a natural strategy is to guess those subkey bits which can lead to a large filter. If each subkey cell is equally profitable (*e.g.*, the attack on `Serpent` in Section 4.1), one can find by hand the subkey k'_b and k'_f to be guessed in the key recovery, so that the time complexity is minimized. However, it is not the case for many ciphers. For certain ciphers, not only the subkey cells are not equally profitable, but also the subkey cells are closely related through the key schedule. Finding the best parameters for them by hand is challenging. Moreover, given a set of parameters that permit an efficient key recovery, one may wonder whether it is optimal or not. Therefore, optimal rectangle attacks are possible only when the above key recovery algorithm is fed with a set of proper parameters.

3.3 Framework for Finding the Best Attacking Parameters

In this subsection, we present a framework which acts as a complement of our new key recovery algorithm. This framework finds the best attacking parameters for the rectangle attack. When we apply the parameters returned by this framework to our key recovery algorithm, the time complexity of the attack will be minimal.

Specifically, the framework takes as input a boomerang distinguisher with (α, δ, P^2), *i.e.*, the input difference and output difference, and its probability, and extended rounds E_d and E_f, and returns (k'_b, k'_f) and the minimal time complexity. In essence, this is a optimization problem which can be solved with various tools. A similarity can be observed in finding optimal differential/linear trails [SHW+14, SWW21], division property [HLM+20], meet-in-the-middle attack [SSD+18], etc. Therefore, tools like Mixed-Integer Linear Programming (MILP) and SAT which are widely used for solving these previously mentioned problems can be applied as well in this framework. Since we want to keep our framework generic and flexible, we will describe it as a template in a high level language. When it comes to a specific cipher, one can instantiate it and solve it with MILP solvers or SAT solvers.

Our framework has five modules:

Difference propagation. Model the differentials $\alpha' \xleftarrow{E_b^{-1}} \alpha$ and $\delta \xrightarrow{E_f} \delta'$, both of which propagate difference with probability 1. Compute r_b and r_f. Mark the state cell if its difference is fixed.

Value path. Mark the state cells whose values are needed for verifying α difference and δ difference. Alongside, mark the subkey k_b and k_f which are needed for the verification.

Guess-and-determine. Model the relation between the subkey bits and the internal state cells, *i.e.*, when certain subkey bits are guessed, the corresponding internal state cell can be determined. When a internal state cell resulting from some active cells is determined and should have a fixed difference, then a filter is reached. Model the number of filters $r'_b + r'_f$.

Key bridging. [5] Model the relation between subkey bits according to the key schedule algorithm. Model the number of *independent* guessed subkey bits $m'_b + m'_f$.

Objective function. Compute $T_i, i \in [0, 4]$ from $P, n, r_b, r_f, r'_b, r'_f, m'_b$ and m'_f. Set the objective function to $min \sum_0^4 T_i$.

Other constraints can imposed alongside, such as constraints on memory. Given a rectangle distinguisher of a certain cipher, one can follow this framework to build a concrete model dedicated to this cipher and try different E_b and E_f to find a set of best parameters. Key information that can be extracted from these parameters include

- Subkey k'_b and k'_f which will be guessed;
- The number of independent key bits in k'_b and k'_f, *i.e.*, $m'_b + m'_f$;
- The overall time complexity.

Feed these parameter to our key recovery algorithm, the rectangle key recovery will be optimized.

3.4 Extensions

In this subsection, we discuss possible extensions of our rectangle key recovery algorithm presented in Section 3.2.

When $r_b = n$. The algorithm in Section 3.2 applies only when $r_b < n$. However, it can be extended to the case when $r_b = n$ by changing the way of choosing plaintexts, as shown in Appendix A.1

[5] "Key bridging" is borrowed from [DKS10a, DKS15] which originally connects two subkeys separated by several key mixing steps.

The related-key setting. The algorithm in Section 3.2 is specifically targeted at the rectangle attack in the single-key setting. With small modifications, it can be adapted to the related-key setting for ciphers with a linear key schedule, as shown in Appendix A.2. This extension is particularly useful as many block ciphers, especially lightweight ones, employ a linear key schedule, *e.g.*, `SKINNY` [BJK+16b] and `Deoxys-BC` [JNPS16].

Boomerang attack. An attacker can only choose plaintexts in rectangle attacks. However, in boomerang attacks, the attacker is allowed to choose plaintexts and ciphertexts adaptively. With this in mind, we also propose variants of our algorithm dedicated for boomerang attacks. We specifically consider the key recovery for $E = E_d \circ E_b$ and $E = E_f \circ E_d$. The detailed algorithm for key recovery in the former case is given in Appendix A.3, whereas the algorithm for the latter case is presented as follows.

Boomerang key recovery for $E = E_f \circ E_d$. Similarly, we assume there exists a distinguisher of E_d, whose probability is P^2, input difference is α and output difference is δ. E_f is appended to E_d and partial subkey k'_f will be guessed.

1. Construct a set S_0 which is made up of y structures, each of 2^{r_f} ciphertexts. Let $D = y \cdot 2^{r_f}$. Query and collect two sets of data:
$$S_1 = \{(P_1, C_1) | P_1 = E^{-1}(C_1), C_1 \in S_0\},$$
$$S_2 = \{(P_2, C_2) | P_2 = P_1 \oplus \alpha, C_2 = E(P_2), P_1 \in S_1\}.$$
2. Split m'_f-bit k'_f into two parts: $G_L \| G_R$ where G_L has t bits.
3. Guess G_R:
   (a) Initialized a list of key counters for G_L and unguessed key bits of k_f.
   (b) Guess the t-bit G_L:
      i. For each data in S_1, S_2, do partial decryptions under k'_f. Let $C_1^* = Dec_{k'_f}(C_1)$ and $C_2^* = Dec_{k'_f}(C_2)$. Then the set of obtained C_1^* contains $y \cdot 2^{r'_f}$ sub-structures, each of $2^{r_f^*}$ ciphetexts.
      ii. Construct a set as
$$S_{1,2} = \{(P_1, C_1^*, P_2, C_2^*) | P_2 = P_1 \oplus \alpha, C_2^* = Dec_{k'_f}(Enc(P_2))\}.$$
      Insert $S_{1,2}$ into a hash table by $n - r_f^*$ inactive bits of C_1^* and $n - r_f^*$ inactive bits of C_2^*.
      iii. There are $y \cdot 2^{r'_f}$ possible values for the $n - r_f^*$ bits of C_1^* and $2^{n-r_f^*}$ possible values for the $n - r_f^*$ bits of C_2^*. For each index, we pick two distinct entries (P_1, C_1^*, P_2, C_2^*) and (P_3, C_3^*, P_4, C_4^*) to generate the quartet. The number of quartet we will get is
$$\binom{\frac{|S_{1,2}|}{2^{n-r_f^*} \cdot y \cdot 2^{r'_f}}}{2} \cdot 2^{n-r_f^*} \cdot y \cdot 2^{r'_f} = D \cdot 2^{2r_f^* - n - 1}.$$
      iv. Determine the key candidates involved in E_f and increase the corresponding counters. Denote the time complexity for processing one quartet as ϵ.

(c) Select the top $2^{t+m_f-m'_f-h}$ hits in the counters to be the candidates, which delivers a h-bit or higher advantage.
(d) Guess the remaining $k-m_f$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key, where k is the key size.

Data complexity. From y structures, we can form $y \cdot 2^{2r_f-1}$ plaintext pairs. Among them, $y \cdot 2^{r_f-1}$ pairs satisfy δ difference on average. Let s be the expected number of right quartets, so we have $y \cdot 2^{r_f-1} \cdot P^2 = s$, $y = s \cdot 2^{1-r_f}/P^2$ and $D = y \cdot 2^{r_f} = 2s/P^2$. Therefore, the data complexity is $D_B = 2D = 4s/P^2$.

Memory complexity. The memory complexity of is $M = D_B + D + 2^{t+m_f-m'_f}$ to store the data, the set $S_{1,2}$ and the counters.

Time complexity. The time complexity of collecting data is $T_0 = D_B$, the time complexity of doing partial encryption and decryption under guessed key bits is

$$T_1 = 2^{m'_f} \cdot D_B = 2^{m'_f} \cdot 2 \cdot y \cdot 2^{r_f} = s \cdot 2^{m'_f+2}/P^2,$$

the time complexity of generating set S is

$$T_2 = 2^{m'_f} \cdot D = s \cdot 2^{m'_f+1}/P^2,$$

the time complexity of generating and processing quartet candidates is

$$T_3 = 2^{m'_f} \cdot D \cdot 2^{2r^*_f} \cdot 2^{-n} \cdot \epsilon = s \cdot 2^{m'_f+2r_f-2r'_f-n}/P^2,$$

and the time complexity of exhaustive search is $T_4 = 2^{m'_f} \cdot 2^{k-m'_f-h} = 2^{k-h}$, where $h \leq 2^{t+m_f-m'_f}$.

4 Applications

In this section, we apply our new key recovery algorithm to four block ciphers using existing distinguishers: `Serpent`, `CRAFT`, `SKINNY`, and `Deoxys-BC-256`. We find that the best attacking parameters differ significantly from those which were used in previous works and even the number rounds in outer part E_b or E_f is different. Moreover, these new attacking parameters are not covered by the previous key recovery algorithms in many cases. Consequently, improved results on these ciphers can be obtained.

4.1 Application to `Serpent`

We apply our new rectangle key recovery algorithm to `Serpent` [ABK98], which was the first target when the rectangle attack was proposed in 2001 [BDK01]. Serpent is a block cipher which ranked the second in the Advanced Encryption

Standard (AES) finalist. It was an SP-network designed by Ross Anderson, Eli Biham, and Lars Knudsen, which has a block size of 128 bits and supports a key size of 128, 192 or 256 bits. Serpent iterates 32 rounds, and each round $i \in \{0, 1, ..., 31\}$ consists of three operations: key mixing, S-boxes and linear transformation. Suppose B_i represents the internal state before round i, K_i is the 128-bit subkey, and S_i denotes the application of S-box in round i. Let L be the linear transformation. Then the Serpent round function is defined as follows.

$$\begin{aligned} X_i &= B_i \oplus K_i \\ Y_i &= S_i(X_i) \\ B_{i+1} &= L(Y_i), i = 0, \cdots, 30 \\ B_{i+1} &= Y_i \oplus K_{i+1}, i = 31 \end{aligned}$$

The internal state of Serpent can be seen as a 4×32 array, where each row is a 32-bit word. The S-boxes is applied to 4-bit columns. Serpent applies eight different 4-bit S-boxes, and these eight S-boxes are used four times. As our attack does not depend on the order of S-boxes, we omit the details here.

Distinguisher. We use the 8-round rectangle distinguisher of Serpent proposed by Biham et al in [BDK01] to attack 10-round Serpent with E_b and E_f consisting of round 0 and round 9 respectively. The probability of the distinguisher is $2^{-n}P^2 = 2^{-128-120.6}$, and other parameters of the attack are: $n = 128, m_b = r_b = 76, m_f = r_f = 20$.

Recently in [KT22], this distinguisher has been re-evaluated and a more accurate probability of $2^{-128-116.3}$ is reported. For a better comparison, we will mount key recovery attack with both probabilities of the distinguisher.

In the case of Serpent, a 4-bit key guess for an active S-box will lead to a 4-bit inner state filter for a pair of messages. That is, all the key nibbles corresponding to the active S-boxes of the first round and the last round are equivalently good for filtering data.

Parameters and complexities. When we take the old probability, the best guessing parameters are $m'_f = r'_f = 20, m'_b = r'_b = 8$, which means guessing all the k_f and two nibbles of k_b. Note that, this type of guessing is not covered in previous rectangle key recovery algorithms. The complexities are as follows.

- The data complexity is $D = y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{n/2+1}/P = \sqrt{s} \cdot 2^{125.3}$.
- The memory complexity is $M = D + D^2 \cdot 2^{r_f^*-n-1} + 2^{t+m_b+m_f-m'_b-m'_f} = \sqrt{s} \cdot 2^{125.3} + s \cdot 2^{121.6} + 2^{t+68}$.
- The time complexity $T_1 = 2^{m'_b+m'_f} \cdot D = \sqrt{s} \cdot 2^{153.3}$;
- $T_2 = 2^{m'_b+m'_f} \cdot D^2 \cdot 2^{r_f^*-n-1} = s \cdot 2^{149.6}$;
- $T_3 = 2^{m'_b+m'_f} \cdot D^2 \cdot 2^{2r_b^*+2r_f^*-2n-2} \cdot \epsilon = s \cdot 2^{28+250.6+2\times68+0-2\times128-2} \cdot \epsilon = s \cdot 2^{156.6} \cdot \epsilon$;
- $T_4 = 2^{k-h}, h < 68 + t$.

For each of the remaining quartets, it can be processed S-box by S-box, so ϵ takes about $1+2^{-4}+2^{-8}+\cdots+2^{-16*4} = 2^{0.09}$ memory accesses. Set $s = 4$, then the data, and memory complexities of our attack are both $2^{126.3}$. The time complexity besides the brute forcing part includes $2^{154.3}$ partial encryptions/decryptions and $2^{158.69}$ memory accesses. Assume a partial encryptions/decryptions is equivalent to 7 memory accesses as 7 S-boxes are involved. Then it needs $2^{159.11}$ memory accesses in total.

When we take the new probability, the guessing parameters $m'_f = r'_f = 20, m'_b = r'_b = 8$ are still the best. Another choice for these parameters is $m'_f = r'_f = 16, m'_b = r'_b = 12$ which leads to the same time complexity but a slightly higher memory complexity. Thus we choose the former one. Set $s = 4$, then the data, and memory complexities of our attack are both $2^{124.15}$. The time complexity besides the brute forcing part include $2^{152.15}$ partial encryptions/decryptions and $2^{154.39}$ memory accesses, which is about $2^{155.67}$ memory accesses in total.

The comparison with the previous rectangle attacks[6] based on the same distinguisher is presented in Table 2.

Table 2: Comparisons of key recovery attacks on 10-round Serpent where the time is measured by the number of memory accesses.

P^2	m_b, m_f	m'_b, m'_f	Data	Memory	Time	Reference
$2^{-120.6}$	$76, 20$	76,20	$2^{126.8}$	2^{192}	2^{217}	[BDK01]
		0,0	$2^{126.3}$	$2^{126.3}$	$2^{173.8}$	[BDK02]
		8,20	$2^{126.3}$	$2^{126.3}$	$2^{159.11}$	This
$2^{-116.3}$	$76, 20$	8,20	$2^{124.15}$	$2^{124.15}$	$2^{155.67}$	This

4.2 Application to CRAFT

We apply our new rectangle key recovery algorithm to CRAFT in single-key setting and obtain the first 19-round rectangle attack which is one round more than the previous work in [HBS21].

Specification. CRAFT is a lightweight tweakable block cipher which was introduced by Beierle *et al.* [BLMR19]. It supports 64-bit plaintext, 128-bit key, and 64-bit tweak. Its round function is composed of involutory building blocks. The 64-bit input is arranged as a state of 4×4 nibbles. The state is then going through 32 rounds $\mathcal{R}_i, i \in 0, \cdots, 31$, to generate a 64-bit ciphertext. As depicted in Figure 4, each round, excluding the last round, has

[6] In [DQSW22], a rectangle attack on 10-round Serpent was also given. However, the authors seem to mistake m_f, r_f for m_b, r_b. So we do not include their result in Table 2.

five functions, *i.e.*, MixColumn (MC), AddRoundConstants (ARC), AddTweakey (ATK), PermuteNibbles (PN), and S-box (SB). The last round only includes MC, ARC and ATK, i.e., $\mathcal{R}_{31} = ATK_{31} \circ ARC_{31} \circ MC$, while for any $0 \le i \le 30$, $\mathcal{R}_i = SB \circ PN \circ ATK_i \circ ARC_i \circ MC$.

The tweakey schedule of CRAFT is rather simple. Given the secret key $K = K_0 \| K_1$ and the tweak $T \in \{0,1\}^{64}$, where $K_i \in \{0,1\}^{64}$, four round tweakeys $TK_0 = K_0 \oplus T$, $TK_1 = K_1 \oplus T$, $TK_2 = K_0 \oplus Q(T)$ and $TK_3 = K_1 \oplus Q(T)$ are generated, where Q is a nibble-wise permutation. Then at the round $\mathcal{R}_i$, $TK_{i\%4}$ is used as the subtweakey.

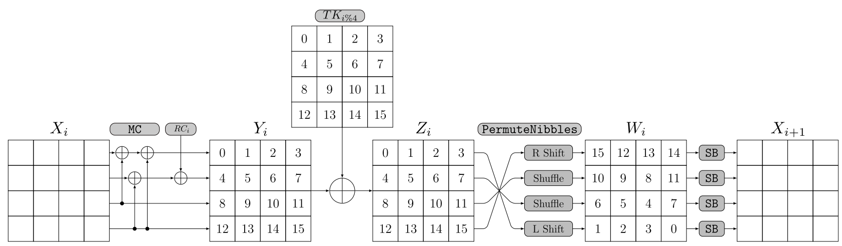

Figure 4: A round of CRAFT

Distinguisher. We use the 14-round rectangle distinguisher of CRAFT proposed by Hosein *et al.* in [HBS21] to attack 19-round CRAFT with 3-round E_b and 2-round E_f, as shown in Figure 5. The probability of the distinguisher is $2^{-n}P^2 = 2^{-64-55.85}$, and other parameters of the attack are: $n = 64, k = 128, m_b = 112, r_b = 60, m_f = r_f = 24$. The first three subtweakeys are TK_0, TK_1, and TK_2, respectively. The last subtweakey is TK_2. Note TK_2 share the same key information with TK_0, and $k_b \cup k_f$ only contain $(16 + 12 + 6 - 6) \times 4 = 112$ information bits.

Parameters and complexities. The best guessing parameters are $m'_b = 32, r'_b = 16, m_f = r'_f = 24$, and $|k'_b \cup k'_f| = 40$, which means guessing 10 cells of k_f and k_b to get 10 cells filters. The key cells to be guessed and the corresponding filters are highlighted with red squares in Figure 5. Note that this type of guessing is not covered in previous rectangle key recovery attacks. The complexities of our new attack are as follows.

- The data complexity is $D = y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{n/2+1}/P = \sqrt{s} \cdot 2^{60.92}$.
- The memory complexity is $M = D + D^2 \cdot 2^{r_f^* - n - 1} + 2^{m_b + m_f - m'_b - m'_f} = \sqrt{s} \cdot 2^{60.92} + s \cdot 2^{56.85} + 2^{t+72}$
- The time complexity $T_1 = 2^{m'_b + m'_f} \cdot D = \sqrt{s} \cdot 2^{100.92}$;
- $T_2 = 2^{m'_b + m'_f} \cdot D^2 \cdot 2^{r_f^* - n - 1} = s \cdot 2^{96.85}$;

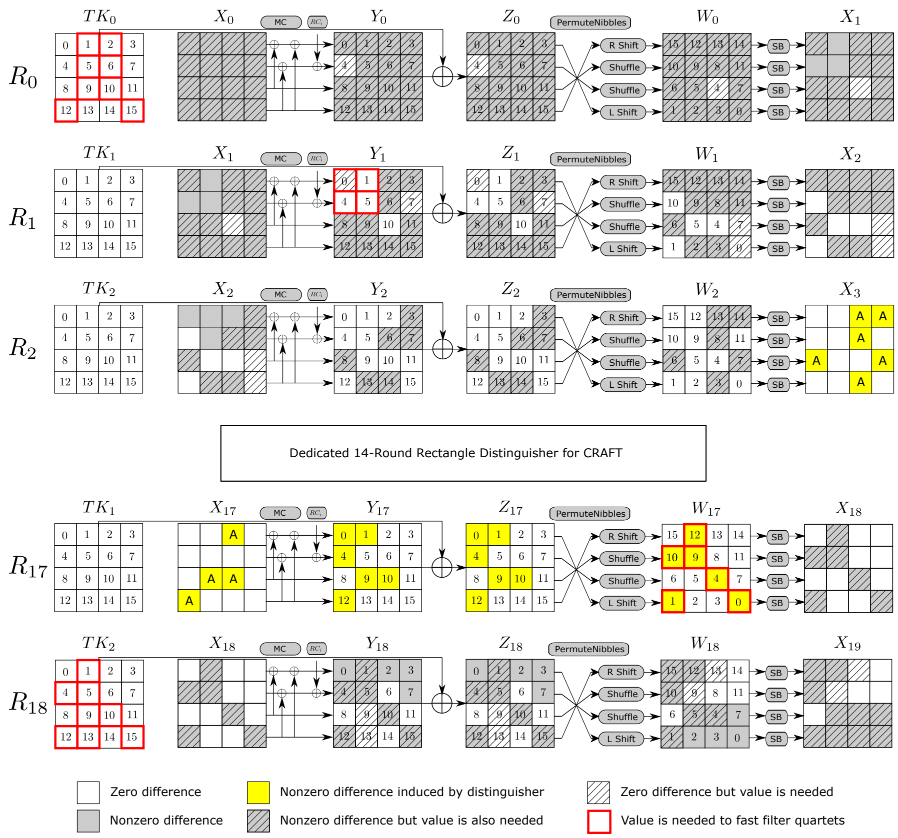

Figure 5: A 19-round key recovery attack against `CRAFT`

- $T_3 = 2^{m'_b+m'_f} \cdot D^2 \cdot 2^{2r_b^*+2r_f^*-2n-2} \cdot \epsilon = s \cdot 2^{40+121.85+2\times 44+0-2\times 64-2} \cdot \epsilon = s \cdot 2^{119.85} \cdot \epsilon$;
- $T_4 = 2^{k-h}, h < t + 72$.

Processing a candidate quartet to retrieve the rest of k_b and can be realized by looking up tables. We pre-compute several tables as illustrated in Table 4. How will these tables be used? For each quartet candidate, we first look up table 1 using known values $(Y_1^i[9], Y_1^i[12]), i = 1, 2, 3, 4$ for $TK_1[9], TK_1[12]$. Each quartet candidate will have one $TK_1[9], TK_1[12]$ on average. Then, look up table 2 using $(Y_0^i[3], X_1^i[2] \oplus X_1^i[10], \Delta X_2^j[1], TK_0[13]), i = 1, 2, 3, 4, j = 1, 3$ and only 2^{-8} of the quartet candidates can find a hit in the table for $TK_0[3], TK_1[2]$. Discard those quartet candidates which can not find a hit in the table. Then look up the next table and so on. Therefore, ϵ is equivalent to about 2 memory accesses which is around $2 \times \frac{1}{16} \times \frac{1}{19} = 2^{-7.24}$ encryption. If we set $s = 1$, $h = 28$ and $t = 0$, then the data, memory and time complexities of our attack are $2^{60.92}$, 2^{72}, and $2^{112.61}$, respectively. The success probability is about 74.59% which is computed by Selçuk's formula [Sel08].

The comparison with the previous rectangle attacks based on the same distinguisher is presented in Table 3.

Table 3: Comparisons of key recovery attacks on CRAFT

P^2	Rounds	m_b, m_f	m'_b, m'_f	Data	Memory	Time	Reference
$2^{-55.85}$	$1+14+3$	$24, 84$	$24, 0$	$2^{60.92}$	2^{84}	$2^{101.7}$	[HBS21]
$2^{-55.85}$	$3+14+2$	$112, 24$	$32, 24$	$2^{60.92}$	2^{72}	$2^{112.61}$	This

4.3 Application to SKINNY

When we apply our new rectangle key recovery algorithm to SKINNY's distinguishers from [DQSW22], better attacks are obtained for three out of four distinguishers, and for the rest one, our attack matches with the one in [DQSW22]. Even though these distinguishers were searched dedicated for the key recovery algorithm in [DQSW22] (named Algorithm 4 in Section 5), we found that the best attacking parameters may be not covered by that key recovery algorithm.

Next, we give the detailed attack on 25-round SKINNY-64-128 and the attacks on 32-round SKINNY-128-384 and 26-round SKINNY-128-256 are postponed to Appendix B.1.

Specification. SKINNY [BJK+16b] is a family of lightweight block ciphers which adopt the substitution-permutation network and elements of the TWEAKEY framework [JNP14]. Members of SKINNY are denoted by SKINNY-n-tk, where $n \in \{64, 128\}$ is the block size and $tk \in \{n, 2n, 3n\}$ is the tweakey size. The internal states of SKINNY are represented as 4×4 arrays of cells with each cell being a nibble in case of $n = 64$ bits and a byte in case of $n = 128$ bits. The tweakey state is seen as a group of z 4×4 arrays, where, $z = tk/n$. The arrays are marked as $TK1$, $(TK1, TK2)$ and $(TK1, TK2, TK3)$ for $z = 1, 2, 3$ respectively.

SKINNY iterates a round function for N_r rounds and each round consists of the following five steps.

1. *SubCells* (SC) - A 4-bit (resp. 8-bit) S-box is applied to all cells when n is 64 (resp. n is 128).
2. *AddConstants* (AC) - This step adds constants to the internal state.
3. *AddRoundTweakey* (ART) - The first two rows of the internal state absorb the first two rows of TK, where $TK = \bigoplus_{i=1}^{z} TK_i$.
4. *ShiftRows* (SR) - Each cell in row j is rotated to the right by j cells.
5. *MixColumns* (MC) - Each column of the internal state is multiplied by matrix M whose branch number is only 2.

Table 4: Precomputation tables for 19-round attack on `CRAFT`, where underlined bytes are used as input and determine the time and memory complexity for building the table. Note that the precomputation table may be built for pairs or quartets. When a table is built for pairs, the filter effect in brackets is for two pairs.

No.	Starting cells	Subtweakey bytes	Bytes deduced	Filter	Pairs or quartets	Time and memory	Filter effect
1	$\underline{Y_1[9]}, \underline{Y_1[12]}, \underline{Y_1'[9]}$	$\underline{TK_1[9]}, \underline{TK_1[12]}$	$X_2[1], X_2[5], Y_1'[12]$	$X_2[1] \oplus X_2'[1] = X_2[5] \oplus X_2'[5]$	Quartets	2^{32}	1
	$(Y_1^i[9], Y_1^i[12]), i = 1,2,3,4 : TK_1[9], TK_1[12]$						
2	$\underline{Y_0[3]}, \underline{\Delta X_2[1]}, \underline{X_1[2] \oplus X_1[10]}, \underline{X_1'[2] \oplus X_1'[10]}, \underline{TK_0[13]}$	$\underline{TK_0[3]}, \underline{TK_1[2]}$	$X_2[13], X_3[2], Y_0'[3]$	$X_2[13] \oplus X_2'[13] = \Delta X_2[1]$ $X_3[2] \oplus X_3'[2] = 0xA$	Quartets	2^{44}, 2^{36}	2^{-8}
	$(Y_0^i[3], X_1^i[2] \oplus X_1^i[10], \Delta X_2^j[1], TK_0[13]), i = 1,2,3,4, j = 1,3 : TK_0[3], TK_1[2]$						
3	$\underline{Y_0[0]}, \underline{Y_0[11]}, \underline{Y_0'[0]}$	$\underline{TK_0[0]}, \underline{TK_0[11]}$	$X_1[7], X_1[15], Y_0'[11]$	$X_1[7] \oplus X_1'[7] = X_1[15] \oplus X_1'[15]$	Quartets	2^{32}	1
	$(Y_0^i[0], Y_0^i[11]), i = 1,2,3,4 : TK_0[0], TK_0[11]$						
4	$\underline{Y_0[8]}, \underline{X_1[14]}, \underline{X_1'[14]}$	$\underline{TK_0[8]}, \underline{TK_1[6]}$	$X_3[6], Y_0'[8]$	$X_3[6] \oplus X_3'[6] = 0xA$	Quartets	2^{32}	1
	$(Y_0^i[8], X_1^i[14]), i = 1,2,3,4 : TK_0[8], TK_1[6]$						
5	$\underline{\Delta X_2[8]}, \underline{Y_1[15]}$	$\underline{TK_1[15]}$	$X_2[0], Y_1'[15]$	$X_2[0] \oplus X_2'[0] = \Delta X_2[8]$	Quartets	2^{20}	2^{-4}
	$(\Delta X_2^j[8], Y_1^i[15]), i = 1,2,3,4, j = 1,3 : TK_1[15]$						
6	$\underline{Y_1[8]}, \underline{Y_1[13]}, \underline{Y_0[7]}, \underline{Y_0[14]}, \underline{X_1[15] \oplus X_1[11]}, \underline{X_1'[15] \oplus X_1'[11]}, \underline{TK_0[6]}$	$\underline{TK_1[3]}, \underline{TK_1[8]}, \underline{TK_1[13]}, \underline{TK_0[7]}, \underline{TK_0[14]}$	$X_2[2], X_2[14], X_3[4], X_3[8], Y_1'[8], Y_1'[3], Y_0'[7], Y_0'[14]$	$X_2[2] \oplus X_2'[2] = X_2[14] \oplus X_2'[14]$, $X_3[3] \oplus X_3'[3] = 0xA$, $X_3[8] \oplus X_3'[8] = 0xA$	Pairs	2^{52}	2^8 (2^{-4})
	$(Y_1^i[8], Y_1^i[13], X_1^i[15] \oplus X_1^i[11], Y_0^i[7], Y_0^i[14], TK_0[6]), i = 1,2 : TK_0[7], TK_0[14], TK_1[3], TK_1[8], TK_1[13]$						
7	$\underline{Y_1[0]}, \underline{Y_1[7]}, \underline{Y_1[11]}, \underline{Y_1[14]}, \underline{TK_0[3]}, \underline{TK_0[7]}$,	$\underline{TK_1[0]}, \underline{TK_1[7]}, \underline{TK_1[11]}, \underline{TK_1[14]}$	$X_3[11], X_3[14], Y_1'[11], Y_1'[14]$	$X_3[11] \oplus X_3'[11] = 0xA$, $X_3[14] \oplus X_3'[14] = 0xA$	Pairs	2^{40}	2^8 (1)
	$(Y_1^j[0], Y_1^j[7], Y_1^i[11], Y_1^i[14], TK_0[3], TK_0[7]), i = 1,2, j = 1 : TK_1[0], TK_1[7], TK_1[11], TK_1[14]$						

The tweakey schedule of `SKINNY` is a linear algorithm. The tk-bit tweakey is first loaded into z 4×4 tweakey states. After each `ART` step, a cell-wised permutation P is applied to each tweakey state, where P is defined as: $P = [9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7]$. Then cells in the first two rows of all tweakey states but TK_1 are individually updated using LFSRs. For complete details of the tweakeys scheduling algorithm, one can refer to [BJK+16b].

Distinguihser of `SKINNY-64-128`. We reuse the 18-round rectangle distinguisher of `SKINNY`-64-128 from [QDW+21, DQSW21] and apply our new rectangle key recovery algorithm to it. As a result, we obtain a new 25-round rectangle attack. The probability of the distinguisher is $2^{-n}P^2 = 2^{-64-55.34} = 2^{-119.34}$. Our key recovery extends the distinguisher by three rounds at the top and four rounds at the bottom, as shown in Figure 6. The parameters for this attack are: $r_b = 8 \times 4 = 32, r_f = 12 \times 4 = 48, m_b = 10 \times 4 = 40$ and $m_f = 21 \times 4 = 84$. Due to the tweakey schedule, we can deduce $SKT_{22}[6, 1, 7, 2]$ from $STK_0[0, 5, 6, 7]$ and $STK_{24}[5, 0, 1, 4]$, and deduce $STK_{21}[6]$ from $STK_1[2]$ and $STK_{23}[5]$. Such that $k_b \cup k_f$ only contain $(31 - 5) \times 4 = 104$ information bits.

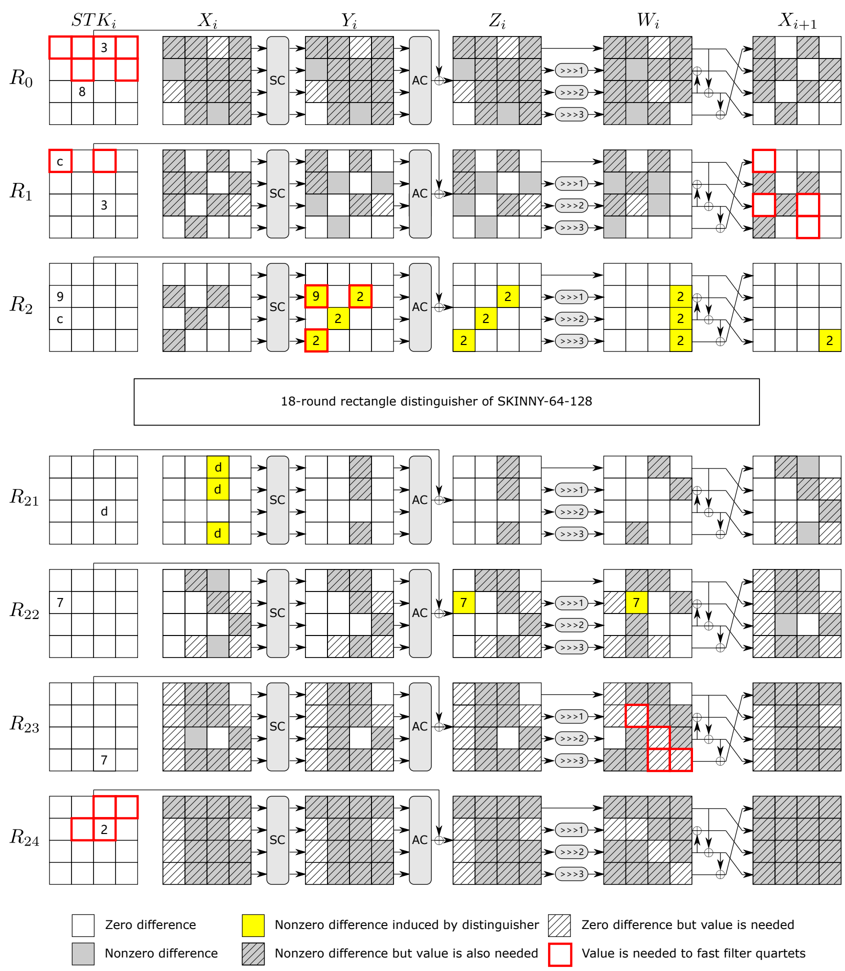

Figure 6: A 25-round key recovery attack against SKINNY-64-128

Parameters and complexities. We apply the related-key version of our new algorithm in Appendix A.2 to the above distinguisher. The best guessing parameters are $m'_b = 32, r'_b = 28$ and $m'_f = r'_f = 16$, which means guessing partial bits of k_b and k_f. This type of guessing is not covered in previous rectangle key recovery attacks. The complexities of our new attack are as follows.

- The data complexity is $D_R = 4 \cdot y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{n/2+2}/P = \sqrt{s} \cdot 2^{61.67}$.
- The memory complexity is $M_R = D_R + D \cdot 2^{r_b^*} + 2^{t+m_b+m_f-m'_b-m'_f} = \sqrt{s} \cdot 2^{61.67} + \sqrt{s} \cdot 2^{63.67} + 2^{56+t}$
- The time complexity $T_1 = 2^{m'_b+m'_f} \cdot D_R = \sqrt{s} \cdot 2^{12\times 4+61.67} = \sqrt{s} \cdot 2^{109.67}$;
- $T_2 = 2^{m'_b+m'_f} \cdot D \cdot 2^{r_b-r'_b} = \sqrt{s} \cdot 2^{12\times 4+59.67+4} = \sqrt{s} \cdot 2^{111.67}$;
- $T_3 = 2^{m'_b+m'_f} \cdot D^2 \cdot 2^{2r_b^*+2r_f^*-2n} \cdot \epsilon = s \cdot 2^{12\times 8+119.34+2\times 4+2\times 32-2\times 64} \cdot \epsilon = s \cdot 2^{111.34} \cdot \epsilon$;
- $T_4 = 2^{128-h}, h < 56 + t$.

Processing a candidate quartet to retrieve the rest of k_b and k_f can be realized by looking up tables. We pre-compute several tables as illustrated in Table 6, so that ϵ is equivalent to about $1 + 1 + 2^4 + 2^4 + 1 = 35$ memory accesses which is around $35 \times \frac{1}{16} \times \frac{1}{25} = 2^{-3.51}$ encryption. If we set $s = 1$, $h = 30$ and $t = 0$, then the data, memory and time complexities of our attack are $2^{61.67}$, $2^{63.67}$, and $2^{110.03}$, respectively. The success probability is about 75.81%.

Table 5: Comparisons of key recovery attacks on SKINNY-64-128

P^2	Rounds	m_b, m_f	m'_b, m'_f	Data	Memory	Time	Reference
$2^{-55.34}$	$2+18+5$	$12, 116$	$12, 40$	$2^{61.67}$	$2^{64.26}$	$2^{118.43}$	[DQSW22]
$2^{-55.34}$	$3+18+4$	$40, 84$	$32, 16$	$2^{61.67}$	$2^{63.67}$	$2^{110.03}$	This

The comparison with the previous rectangle attacks based on the same distinguisher is presented in Table 5.

4.4 Application to Deoxys-BC-256

Due to the relationship between the boomerang attack and rectangle attack, we propose variants of our algorithm dedicated for boomerang attacks. We apply our new algorithm to Deoxys-BC-256 and obtain the first 11-round boomerang attack and an improved 11-round rectangle attack. Next, we give detailed attack on 11-round boomerang attack and the 11-round rectangle attack is postponed to Appendix B.2.

Specification. Deoxys-BC is an AES-based tweakable block cipher [JNPS16], based on the tweakey framework [JNP14]. The Deoxys authenticated encryption scheme makes use of two versions of the cipher as its internal primitive:

Table 6: Precomputation tables for 25-round attack on SKINNY-64-128, where underlined bytes are used as input and determine the time and memory complexity for building the table. Note that the precomputation table may be built for pairs or quartets. When a table is built for pairs, the filter effect in brackets is for two pairs.

No.	Starting cells	Subtweakey bytes	Bytes deduced	Filter	Pairs or quartets	Time and memory	Filter effect
1	$\underline{Z_{24}[1]}$, $\underline{X_{24}[6]}$ $\underline{X_{24}[13]}$, $\underline{X'_{24}[13]}$	$\underline{STK_{24}[1]}$, $\underline{STK_{23}[2]}$	$X_{23}[2]$, $X_{23}[14]$, $Z'_{24}[1]$, $X'_{24}[6]$	$X_{23}[2] \oplus X'_{23}[2] = X_{23}[14] \oplus \underline{X'_{23}[14]}$	Quartets	2^{48}	1
	$(Z^i_{24}[1], X^i_{24}[13], X^i_{24}[6]), i = 1, 2, 3, 4 : STK_{24}[1], STK_{23}[2]$						
2	$\underline{Z_{24}[7]}$, $\underline{X_{24}[11] \oplus X_{24}[15]}$, $\underline{X'_{24}[11] \oplus X'_{24}[15]}$ $\underline{\Delta X_{23}[14]}$	$\underline{STK_{24}[7]}$, $\underline{STK_{23}[6]}$	$X_{23}[6]$, $Z'_{24}[7]$	$X_{23}[6] \oplus X'_{23}[6] = \Delta X_{23}[14]$	Quartets	2^{40}	1
	$(Z^i_{24}[7], X^i_{24}[11] \oplus X^i_{24}[15], \Delta X^j_{23}[14]), i = 1, 2, 3, 4, j = 1, 3 : STK_{24}[7], STK_{23}[6]$						
3	$\underline{Z_{23}[5]}$, $\underline{X_{23}[9]}$, $\underline{Z_{24}[0]}$, $\underline{X_{24}[12]}$, $\underline{X'_{23}[9]}$, $\underline{X'_{24}[12]}$	$\underline{STK_{24}[0]}$, $\underline{STK_{23}[5]}$	$X_{23}[5]$, $X_{23}[12]$ $Z'_{23}[5]$, $Z'_{24}[0]$,	$X_{23}[5] \oplus X'_{23}[5] \oplus X_{23}[9] \oplus X'_{23}[9] \oplus X_{23}[13] \oplus \underline{X'_{23}[13]} = 0x7$	Pairs	2^{36}	2^4 (1)
	$(Z^i_{24}[0], X^i_{24}[12], Z^i_{23}[5], X^i_{23}[9]), i = 1, 2 : STK_{24}[0], STK_{23}[5]$						
4	$\underline{Z_{22}[2]}$, $\underline{X_{23}[13]}$, $\underline{Z_{23}[1]}$, $\underline{X'_{23}[13]}$, $\underline{STK_{22}[2]}$	$\underline{STK_{23}[1]}$,	$X_{22}[2]$, $Z'_{22}[2]$, $Z'_{23}[1]$	$X_{22}[2] \oplus X'_{22}[2] = \underline{\Delta X_{22}[14]}$	Quartets	2^{48}	2^{-4}
	$(Z^i_{23}[1], Z^i_{22}[2], X^i_{23}[13], STK_{22}[2]), i = 1, 2, 3, 4 : STK_{23}[1]$						
5	$\underline{Z_{24}[4]}$, $\underline{Z_{23}[0]}$, $\underline{Z_{22}[1]}$, $\underline{X_{23}[12]}$ $\underline{STK_{22}[1]}$	$\underline{STK_{24}[4]}$, $\underline{STK_{23}[0]}$	$X_{21}[14]$, $Z'_{22}[1]$	$X_{21}[14] \oplus X'_{21}[14] = 0xd$	Quartets	2^{44}	1
	$(Z^i_{22}[1], Z^j_{24}[4], Z^j_{23}[0], X^j_{23}[12], STK_{22}[1]), i = 1, 2, 3, 4, j = 1, 3 : STK_{24}[4], STK_{23}[0]$						
6	$\underline{Z_{23}[7]}$, $\underline{\Delta X_{22}[14]}$ $\underline{X_{23}[11] \oplus X_{23}[15]}$ $\underline{STK_{22}[6]}$	$\underline{STK_{23}[7]}$	$X_{22}[6]$, $X'_{23}[11] \oplus X'_{23}[15]$	$X_{22}[6] \oplus X'_{22}[6] = \Delta X_{22}[14]$	Quartets	2^{32}	2^{-4}
	$(X^i_{23}[11] \oplus X^i_{23}[15], Z^j_{23}[7], \Delta X^j_{22}[14], STK_{22}[6]), i = 1, 2, 3, 4, j = 1, 3 : STK_{23}[7]$						
7	$\underline{Z_{21}[2]}$, $\underline{Z_{23}[4]}$, $\underline{X_{22}[11] \oplus X_{22}[15]}$, $\underline{X_{23}[13]}$, $\underline{STK_{22}[7]}$, $\underline{STK_{21}[6]}$	$\underline{STK_{23}[4]}$ $\underline{STK_{21}[2]}$	$X_{21}[2]$, $X_{21}[6]$, $X'_{22}[11] \oplus X'_{22}[15]$, $Z'_{21}[2]$	$X_{21}[6] \oplus X'_{21}[6] = 0xd$ $X_{21}[2] \oplus X'_{21}[2] = 0xd$	Quartets	2^{48}	2^{-8}
	$(Z^i_{21}[2], X^i_{22}[11] \oplus X^i_{22}[15], X^j_{23}[13], Z^j_{23}[4], STK_{22}[7], STK_{21}[6]), i = 1, 2, 3, 4, J = 1, 3 : STK_{23}[4], STK_{21}[2]$						
8	$\underline{Y_0[6]}$, $\underline{W_0[11]}$ $\underline{Y_1[4]}$	$\underline{STK_0[6]}$, $\underline{STK_1[4]}$	$W_0[7]$, $X_1[11]$ $W_1[9]$, $Y'_1[4]$	$Y_2[9] \oplus Y'_2[9] = 0x2$	Quartets	2^{32}	1
	$(Y^i_1[4], Y^j_0[6], W^j_0[11]), i = 1, 2, 3, 4, j = 1, 3 : STK_0[6], STK_1[4]$						

Deoxys-BC-256 and Deoxys-BC-384. Both versions are ad-hoc 128-bit tweakable block ciphers which besides the two standard inputs, a plaintext P (or a ciphertext C) and a key K, also take an additional input called a *tweak* T.

The concatenation of the key and tweak states is called the *tweakey* state. For `Deoxys-BC-256` the tweakey size is 256 bits.

`Deoxys-BC` is an `AES`-like design, *i.e.*, it is an iterative substitution-permutation network (SPN) that transforms the initial plaintext (viewed as a 4×4 matrix of bytes) using the `AES` round function, with the main differences with `AES` being the number of rounds and the round subkeys that are used every round. `Deoxys-BC-256` has 14 rounds.

Similarly to the `AES`, one round of `Deoxys-BC` has the following four transformations applied to the internal state in the order specified below:

- AddRoundTweakey – XOR the 128-bit round subtweakey to the internal state.
- SubBytes – Apply the 8-bit `AES` S-box to each of the 16 bytes of the internal state.
- ShiftRows – Rotate the 4-byte i-th row left by $\rho[i]$ positions, where $\rho = (0, 1, 2, 3)$.
- MixColumns – Multiply the internal state by the 4×4 constant MDS matrix of `AES`.

After the last round, a final AddRoundTweakey operation is performed to produce the ciphertext.

We denote the concatenation of the key K and the tweak T as KT, i.e. $KT = K||T$. The *tweakey* state is then divided into 128-bit words. More precisely, in `Deoxys-BC-256` the size of KT is 256 bits with the first (most significant) 128 bits of KT being denoted W_2; the second word is denoted by W_1. Finally, we denote by STK_i the 128-bit *subtweakey* that is added to the state at round i during the AddRoundTweakey operation. For `Deoxys-BC-256`, a subtweakey is defined as $STK_i = TK_i^1 \oplus TK_i^2 \oplus RC_i$. The 128-bit words TK_i^1, TK_i^2 are outputs produced by a special *tweakey schedule* algorithm, initialised with $TK_0^1 = W_1$ and $TK_0^2 = W_2$ for `Deoxys-BC-256`. The tweakey schedule algorithm is defined as $TK_{i+1}^1 = h(TK_i^1)$, $TK_{i+1}^2 = h(LFSR_2(TK_i^2))$, where the byte permutation h is defined as

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 1 & 6 & 11 & 12 & 5 & 10 & 15 & 0 & 9 & 14 & 3 & 4 & 13 & 2 & 7 & 8 \end{pmatrix},$$

with the 16 bytes of a 128-bit tweakey word numbered by the usual `AES` byte ordering.

Boomerang Attack. We reuse the 9-round boomerang distinguisher of `Deoxys-BC`-256 proposed by Cid *et al.* [CHP+17, WP19] to attack 11-round boomerang `Deoxys-BC`-256 with 2-round E_f, as shown in Figure 7. The probability of the distinguisher is $P^2 = 2^{-120.4}$, and other parameteres are: $n = 128, k = 256, m_b = r_b = 0, m_f = (16 + 10) \times 8 = 208, r_f = 16 \times 8 = 128$.

The best guessing parameters are $m_f' = 12 \times 8 = 96$ and $r_f' = 8 \times 8 = 64$, which means guessing 8 bytes of k_f. The complexities of our new attack are as follows.

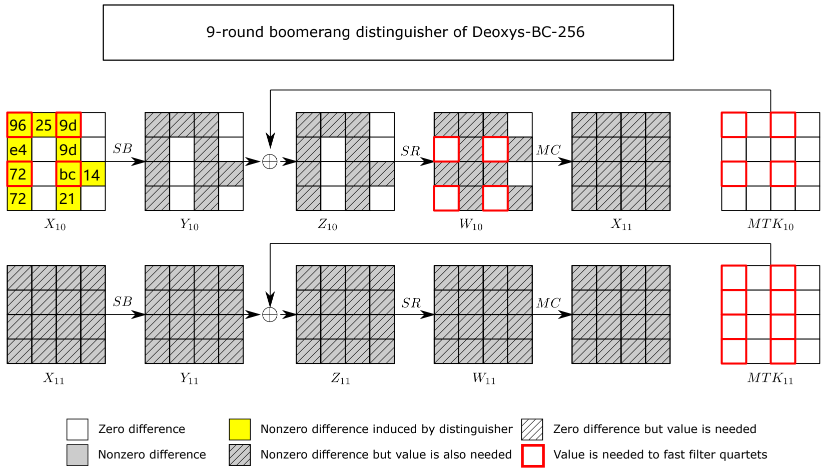

Figure 7: Rectangle/Boomerang attack on 11-round reduced `Deoxys-BC`-256

- The data complexity is $D_{RB} = 4s/P^2 = s \cdot 2^{122.4}$.
- The memory complexity is $M_{RB} = D_{RB} + D + 2^{m_f - m'_f + t} = s \cdot 2^{122.4} + s \cdot 2^{120.4} + 2^{112+t}$.
- The time complexity $T_1 = 2^{m'_f} \cdot D_{RB} = 2^{96} \cdot s \cdot 2^{122.4} = s \cdot 2^{218.4}$;
- $T_2 = 2^{m'_f} \cdot D = s \cdot 2^{216.4}$;
- $T_3 = 2^{m'_f} \cdot D \cdot 2^{2(r_f - r'_f)} \cdot 2^{-n} \cdot \epsilon = s \cdot 2^{96+120.4+2\times 64-128} \cdot \epsilon = 2^{212.4} \cdot \epsilon$;
- $T_4 = 2^{256-h}, h < 112 + t$.

We consider the equivalent round subtweakey $MTK_i = SR^{-1} \circ MC^{-1}(STK_i)$ in round i. To process a candidate quartet to retrieve the rest of k_f,we prepare some tables as illustrated in Table 8. So that ϵ is equivalent to about 1 memory accesses which is around $1 \times \frac{1}{16} \times \frac{1}{11} = 2^{-7.45}$ encryption. If we set $s = 1, h = 40$ and $t = 0$, then the data, memory and time complexities of our attack are $2^{122.4}$, 2^{128}, $2^{218.65}$, respectively. The success probability is about 68.89%. The comparison with the previous boomerang attacks is presented in Table 7.

Table 7: Comparisons of key recovery attacks on `Deoxys-BC`-256

P^2	Rounds	m_b, m_f	m'_b, m'_f	Data	Memory	Time	Reference
$2^{-96.4}$	10	0,88	0,0	$2^{98.4}$	2^{88}	$2^{249.9}$	[ZDJ19a]
$2^{-120.4}$	11	0,208	0,96	$2^{122.4}$	2^{128}	$2^{218.65}$	This

Table 8: Precomputation tables for 11-round attack on `Deoxys-BC`-256, where underlined bytes are used as input and determine the time and memory complexity for building the table.

No.	Starting cells	Subtweakey bytes	Bytes deduced	Filter	Pairs or quartets	Time and memory	Filter effect
1	$\underline{X_{10}[1,11]}$ $\underline{W_{10}[12,14]}$	$\underline{MTK_{11}[12\sim 15]}$ $\underline{MTK_{10}[1,11]}$	$Z_{11}[12\sim 15]$ $Z'_{11}[12\sim 15]$	$\varDelta X_{10}[1,6,11,12] = 0xe4\|00\|21\|00$	Quartets	2^{112}	2^{-16}
	$(Z^i_{11}[12\sim 15]), i=1,2,3,4: MTK_{11}[12\sim 15], MTK_{10}[1,6]$						
2	$\underline{X_{10}[9,14,3,14]}$	$MTK_{11}[4\sim 7]$, $\underline{MTK_{10}[4,}$ $\underline{9,14,3]}$	$Z_{11}[4\sim 7]$, $Z'_{11}[4\sim 7]$	$\varDelta X_{10}[4,9,14,3] = 0x25\|0x9d\|0x14\|72$	Quartets	2^{128}	1
	$(Z^i_{11}[4,5,6,7]), i=1,2,3,4: MTK_{11}[4,5,6,7], MTK_{10}[4,9,14,3]$						

5 Comparison with Related Works

Rectangle key recovery algorithms in previous works. The rectangle attack was proposed by Biham, Dunkelman, and Keller in [BDK01] and has been applied to `Serpent` [ABK98]. The key recovery algorithm used for attacking `Serpent` is rewritten in Appendix C.1. Later, the same authors introduced a new rectangle key recovery algorithm in [BDK02] which improves the result on `Serpent` by reducing the time complexity. Since then, no much progress has been made until Zhao *et al.* proposed a new key recovery algorithm in [ZDM+20] which originally works for ciphers with a linear key schedule in the related-key setting, but it can be converted to the single-key setting trivially. Such an algorithm, when applied to `SKINNY`, outperforms the two previous key recovery algorithms. However, the algorithm presented in a very recent work [DQSW22] makes a step further on improving rectangle attacks on `SKINNY`. For convenience, we call these four rectangle key recovery algorithm in a chronological order by Algorithm 1, Algorithm 2, Algorithm 3, and Algorithm 4, respectively. Details of these algorithms can be found in Appendix C. As concluded in [DQSW22], these algorithms seem independent and perform differently for different parameters. Given a rectangle distinguisher, one can pick the algorithm with lowest complexity from them.

Similarities between our algorithm and the previous algorithms. Our new algorithm reuses some techniques of the previous algorithms.

- Like Algorithm 2, we exploit hash tables when generating pairs and quartets. It costs a certain amount of memory (not necessarily increases the overall memory complexity), but the time complexity is lowered.
- When constructing quartets, we apply the filters on both pairs simultaneously with the help of hash tables. This is also a strategy to trade memory with time which has been used in Algorithm 3 and 4.
- When processing a quartet, we make use of pre-computated tables so that the term ϵ appearing in the time complexity is as small as possible. This has been suggested in Algorithm 2 and we develop this technique in a more practical way.

Our new algorithm unifies all the previous rectangle key recovery algorithms. All the previous four algorithms are distinct from each other by the the number of guessed key bits. Figure 8 illustrates the comparison of our algorithm with the four previous algorithms.

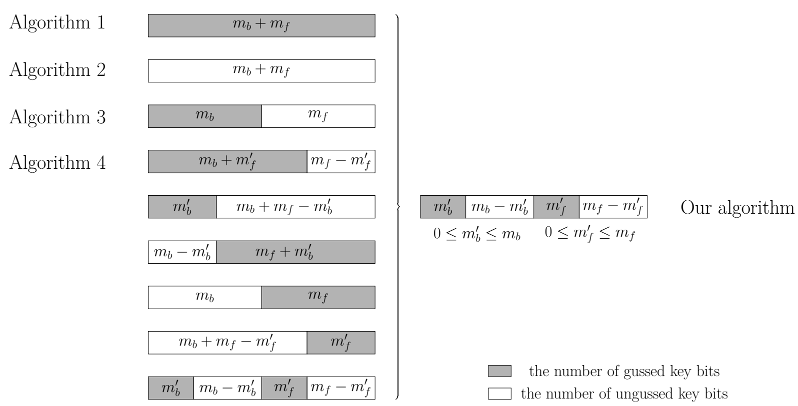

Figure 8: Diagram of guessed key for different algorithms

Specifically, Algorithm 1 guesses the full $(m_b + m_f)$-bit subkey; the main refinement of Algorithm 2 is to generate quartets with birthday paradox without guessing key bits involved in E_b and E_f; Algorithm 3 guesses the m_b-bit key bits involved in E_b to generate quartets; Algorithm 4 extended Algorithm 3 by guessing additional key bits in E_f and exploiting the inner state bits as fast filters.

Our new algorithm supports any number of guessed key bits. Hence, it not only covers all the cases considered by the four previous algorithms, but also includes five types of new cases (see Figure 8).

Any of the previous four algorithms is a special case of our algorithm. We summarize the complexities of different algorithms in Table 9 using notations in this paper. Note the data complexity D remains the same and all the algorithms have to store the data and the subkey counters[7]. Some algorithm may need some extra memory. Therefore, we mainly focus on the comparison of the time complexity and the extra memory complexity.

From complexities listed in Table 9, we can see that Algorithm 1 to 4 are special cases of our algorithm by substituting the corresponding parameters–the exact number of guessed subkey bits and the number of resulted filters–for $m'_b + m'_f$ and r'_b, r'_f in our formulas shown in the last big row of Table 9. Note $r^*_b = r_b - r'_b, r^*_f = r_f - r'_f$. More specifically,

[7] The key counters can be set flexibly. Thus the memory cost for them is elastic.

1. When replacing $m'_b = m_b, m'_f = m_f$ and setting $t = m_b + m_f$, we have Algorithm 1. Since $r^*_b = r^*_f = 0$, the time complexities T_2, T_3 disappear or can be neglected.
2. Algorithm 2 is the case of our algorithm with $m'_b = m'_f = 0, t = 0$ which constructs pairs on the bottom side for ciphertexts.
3. Algorithm 3 is the case of our algorithm with $m'_b = m_b, m'_f = 0$ which constructs pairs on the top side for plaintexts.
4. Algorithm 4 is the case of our algorithm with $m_b + m'_f$ guessed key bits which constructs pairs on the top side for plaintexts.

Table 9: Comparisons of different rectangle key recovery algorithms

Alg.	#Guessed bits	Extra memory	Time
1	$m_b + m_f$	0	$T_1 = 2^{m_b+m_f} \cdot D$
2	0	0	$T_2 = D^2 \cdot 2^{r_f-n-1} = \frac{D}{2} \cdot 2^{r_f-\mu}$ $T_3 = D^2 \cdot 2^{2r_b+2r_f-2n-2} \cdot \epsilon_2$
3	m_b	$\frac{D}{2}$	$T_1 = 2^{m_b} \cdot D$ $T_2 = 2^{m_b} \cdot \frac{D}{2}$ $T_2 = 2^{m_b} \cdot D^2 \cdot 2^{2r_f-2n-2} \cdot \epsilon_3$
4	$m_b + m'_f$	$\frac{D}{2}$	$T_1 = 2^{m_b+m'_f} \cdot D$ $T_2 = 2^{m_b+m'_f} \cdot \frac{D}{2}$ $T_2 = 2^{m_b+m'_f} \cdot D^2 \cdot 2^{2r^*_f-2n-2} \cdot \epsilon_4$
This	$m'_b + m'_f$	$\frac{D}{2} \cdot min\{2^{r^*_b}, 2^{r^*_f-\mu}\}$	$T_1 = 2^{m'_b+m'_f} \cdot D$ $T_2 = 2^{m'_b+m'_f} \cdot \frac{D}{2} \cdot min\{2^{r^*_b}, 2^{r^*_f-\mu}\}$ $T_3 = 2^{m'_b+m'_f} \cdot D^2 \cdot 2^{2r^*_b+2r^*_f-2n-2} \cdot \epsilon$

Application to concrete ciphers. Previously, the four previous key recover algorithms are treated as separate ones. Given a rectangle distinguisher, one can compute the complexities for different algorithms and pick the algorithm with the lowest complexity. Now, with the new algorithm, we can work with this one only and the best parameters that allow to minimize the time complexity may likely lie outside the cases covered by the four previous algorithms. Section 4 includes a series of such examples.

6 Concluding Remarks

In this paper, we propose a unified and generic rectangle key recovery algorithm as well as a framework for automatically finding the best attacking parameters. Combining both, we can find the optimal rectangle attack in terms of time complexity for a given distinguisher. We also extend the new algorithm to other related attacks, such as rectangle attacks in the related-key setting for ciphers with a linear key schedule and boomerang attacks in both the single-key and related-key setting. Applications to block ciphers `Serpent`, `CRAFT`, `SKINNY` and `Deoxys-BC`-256 show that the best rectangle or boomerang attacks are missed by the previous key recovery algorithms in many cases. Thus, better attacks can be obtained. Also, it is likely that previous rectangle attacks can be improved to some extent using the new key recovery algorithm.

Future works. In this paper, we only apply the new rectangle key recovery algorithm to SPN ciphers. However, it should be noted that it is also applicable to Feistel ciphers. Our new key recovery algorithm is generic and does not exploit any property of the S-box as studied in [BCF+21]. It would be a potential future work to exploit properties of the S-box and find more fine-grained parameters for the new algorithm. To search rectangle distinguishers with the new key recovery algorithm taken into account is another topic of interest.

References

ABK98. Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A proposal for the advanced encryption standard. *NIST AES Proposal*, 174:1–23, 1998.

BCF+21. Marek Broll, Federico Canale, Antonio Flórez-Gutiérrez, Gregor Leander, and María Naya-Plasencia. Generic framework for key-guessing improvements. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 453–483. Springer, 2021.

BDK01. Eli Biham, Orr Dunkelman, and Nathan Keller. The rectangle attack—rectangling the Serpent. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 340–357. Springer, 2001.

BDK02. Eli Biham, Orr Dunkelman, and Nathan Keller. New results on boomerang and rectangle attacks. In *International Workshop on Fast Software Encryption*, pages 1–16. Springer, 2002.

BDK05. Eli Biham, Orr Dunkelman, and Nathan Keller. Related-key boomerang and rectangle attacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 507–525. Springer, 2005.

BJK+16a. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In *Annual International Cryptology Conference*, pages 123–153. Springer, 2016.

BJK+16b. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 123–153. Springer, 2016.

BK09. Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In *International conference on the theory and application of cryptology and information security*, pages 1–18. Springer, 2009.

BLMR19. Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. CRAFT: lightweight tweakable block cipher with efficient protection against DFA attacks. *IACR Trans. Symmetric Cryptol.*, 2019(1):5–45, 2019.

BS91. Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

CHP+17. Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. A security analysis of Deoxys and its internal tweakable block ciphers. *IACR Trans. Symmetric Cryptol.*, 2017(3):73–107, 2017.

CHP+18. Carlos Cid, Tao Huang, Thomas Peyrin, Yu Sasaki, and Ling Song. Boomerang connectivity table: a new cryptanalysis tool. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 683–714. Springer, 2018.

DKS10a. Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved single-key attacks on 8-round AES-192 and AES-256. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 158–176. Springer, 2010.

DKS10b. Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony. In *Annual cryptology conference*, pages 393–410. Springer, 2010.

DKS14. Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time related-key attack on the KASUMI cryptosystem used in GSM and 3G telephony. *Journal of cryptology*, 27(4):824–849, 2014.

DKS15. Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved single-key attacks on 8-round AES-192 and AES-256. *J. Cryptol.*, 28(3):397–422, 2015.

DQSW21. Xiaoyang Dong, Lingyue Qin, Siwei Sun, and Xiaoyun Wang. Key guessing strategies for linear key-schedule algorithms in rectangle attacks. *IACR Cryptol. ePrint Arch.*, page 856, 2021.

DQSW22. Xiaoyang Dong, Lingyue Qin, Siwei Sun, and Xiaoyun Wang. Key guessing strategies for linear key-schedule algorithms in rectangle attacks. *To appear at EUROCRYPT 2022*, 2022.

HBS21. Hosein Hadipour, Nasour Bagheri, and Ling Song. Improved rectangle attacks on SKINNY and CRAFT. *IACR Transactions on Symmetric Cryptology*, pages 140–198, 2021.

HLM+20. Yonglin Hao, Gregor Leander, Willi Meier, Yosuke Todo, and Qingju Wang. Modeling for three-subset division property without unknown subset - improved cube attacks against Trivium and Grain-128AEAD. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 466–495. Springer, 2020.

JNP14. Jérémy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International*

Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II, volume 8874 of *Lecture Notes in Computer Science*, pages 274–288. Springer, 2014.

JNPS16. Jérémy Jean, Ivica Nikolic, Thomas Peyrin, and Yannick Seurin. Deoxys v1. 41. *Submitted to CAESAR*, 124, 2016.

KKS00. John Kelsey, Tadayoshi Kohno, and Bruce Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In Bruce Schneier, editor, *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*, volume 1978 of *Lecture Notes in Computer Science*, pages 75–93. Springer, 2000.

KT22. Andreas B. Kidmose and Tyge Tiessen. A formal analysis of boomerang probabilities. *IACR Transactions on Symmetric Cryptology*, 2022(1):88–109, Mar. 2022.

LGS17. Guozhen Liu, Mohona Ghosh, and Ling Song. Security analysis of SKINNY under related-tweakey settings. *IACR Trans. Symmetric Cryptol.*, 2017(3):37–72, 2017.

Mur11. Sean Murphy. The return of the cryptographic boomerang. *IEEE Transactions on Information Theory*, 57(4):2517–2521, 2011.

QDW+21. Lingyue Qin, Xiaoyang Dong, Xiaoyun Wang, Keting Jia, and Yunwen Liu. Automated search oriented to key recovery on ciphers with linear key schedule applications to boomerangs in SKINNY and forkskinny. *IACR Trans. Symmetric Cryptol.*, 2021(2):249–291, 2021.

Sel08. Ali Aydın Selçuk. On probability of success in linear and differential cryptanalysis. *Journal of Cryptology*, 21(1):131–147, 2008.

SHW+14. Siwei Sun, Lei Hu, Peng Wang, Kexin Qiao, Xiaoshuang Ma, and Ling Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 158–178. Springer, 2014.

SQH19. Ling Song, Xianrui Qin, and Lei Hu. Boomerang connectivity table revisited: Application to SKINNY and AES. *IACR Trans. Symmetric Cryptol.*, 2019(1):118–141, 2019.

SSD+18. Danping Shi, Siwei Sun, Patrick Derbez, Yosuke Todo, Bing Sun, and Lei Hu. Programming the demirci-selçuk meet-in-the-middle attack with constraints. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 3–34. Springer, 2018.

SWW21. Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the search of differential and linear characteristics with the SAT method. *IACR Trans. Symmetric Cryptol.*, 2021(1):269–315, 2021.

Wag99. David A. Wagner. The boomerang attack. In Lars R. Knudsen, editor, *Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy, March 24-26, 1999, Proceedings*, volume 1636 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 1999.

WP19. Haoyang Wang and Thomas Peyrin. Boomerang switch in multiple rounds. application to AES variants and deoxys. *IACR Trans. Symmetric Cryptol.*, 2019(1):142–169, 2019.

ZDJ19a. Boxin Zhao, Xiaoyang Dong, and Keting Jia. New related-tweakey boomerang and rectangle attacks on Deoxys-BC including BDT effect. *IACR Trans. Symmetric Cryptol.*, 2019(3):121–151, 2019.

ZDJ19b. Boxin Zhao, Xiaoyang Dong, and Keting Jia. New related-tweakey boomerang and rectangle attacks on Deoxys-BC including BDT effect. *IACR Transactions on Symmetric Cryptology*, pages 121–151, 2019.

ZDM+20. Boxin Zhao, Xiaoyang Dong, Willi Meier, Keting Jia, and Gaoli Wang. Generalized related-key rectangle attacks on block ciphers with linear key schedule: applications to SKINNY and GIFT. *Designs, Codes and Cryptography*, 88(6):1103–1126, 2020.

A Extensions of our Generic Framework

A.1 Rectangle Attacks in the Single-Key Setting with $r_b = n$

When $r_b = n$, we collect $D = 2^x$ plaintext-ciphertext pairs, from which $(\frac{1}{2}D^2)^2 \times 2^{-2n}$ quartets will satisfy the α difference. Let s be the number of expected right quartets. Then, $D = (4s)^{1/4}2^{3n/4}/\sqrt{P}$ and $x = \log_2 D$.

1. Collect and store $D = 2^x$ plaintexts.
2. Split $(m'_b + m'_f)$-bit $k'_b \| k'_f$ into two parts: $G_L \| G_R$ where G_L has t bits.
3. Guess G_R:
   (a) Initialized a list of key counters for G_L and unguessed key bits of k_b, k_f.
   (b) Guess the t-bit G_L:
      i. For each message, partially encrypt P_1 under the k'_b and partially decrypt C_1 under the k'_f. Let $P_1^* = Enc_{k'_b}(P_1)$ and $C_1^* = Dec_{k'_f}(C_1)$. Suppose the state bits that k'_b corresponds to are traversed in the collected data set. Then, we will get $2^{r'_b}$ sub-sets each of $2^{x-r'_b}$ plaintexts.
      ii. Let $2^{-\mu} = D \cdot 2^{-n}$ (*i.e.*, $\mu = n - x$), $r_f^* = r_f - r'_f$ and $r_b^* = r_b - r'_b = n - r'_b$. If $r_b^* < r_f^*$, it turns to step (A); otherwise, it turns to step (D).
         A. Insert all the obtained (P_1^*, C_1^*) into a hash table according to $n - r_b^*$ bits of P_1^*. Then construct a set as $S = \{(P_1^*, C_1^*, P_2^*, C_2^*)$: P_1^* and P_2^* have difference in r_b^* bits$\}$. The size of S is $2^{r'_b} \cdot 2^{2(x-r'_b)-1} = D \cdot 2^{x-r'_b-1} = D \cdot 2^{r_b^*-\mu-1}$.
         B. Insert S into a hash table by $n - r_f^*$ inactive bits of C_1^* and $n - r_f^*$ inactive bits of C_2^*.
         C. With each $2(n-r_f^*)$-bit index, we pick two distinct $(P_1^*, C_1^*, P_2^*, C_2^*)$, $(P_3^*, C_3^*, P_4^*, C_4^*)$ to generate the quartet. We will get
$$2 \cdot \binom{\frac{|S|}{2^{2(n-r_f^*)}}}{2} \cdot 2^{2(n-r_f^*)} = D^2 \cdot 2^{2r_b^*} \cdot 2^{2r_f^*} \cdot 2^{-2n-2\mu-2}$$
quartets. Then go to step (iii).

D. Insert all the obtained (P_1^*, C_1^*) into a hash table according to $n - r_f^*$ bits of P_1^*. Then construct a set as $S = \{(P_1^*, C_1^*, P_3^*, C_3^*)$: C_1^* and C_3^* are colliding in $n - (r_f - r'_f) = n - r_f^*$ bits$\}$. The size of S is $D^2 \cdot 2^{r_f - r'_f - n - 1} = D \cdot 2^{r_f - r'_f - 1 - \mu} = D \cdot 2^{r_f^* - 1 - \mu}$.

E. Insert S into a hash table by r'_b inactive bits of P_1^* and r'_b inactive bits of P_3^*.

F. For each index, we pick two distinct entries $(P_1^*, C_1^*, P_3^*, C_3^*)$, $(P_2^*, C_2^*, P_4^*, C_4^*)$ to generate the quartet. We will get

$$2 \cdot \binom{\frac{|S|}{2^{2r'_b}}}{2} \cdot 2^{2r'_b} = D^2 \cdot 2^{2r_b^*} \cdot 2^{2r_f^*} \cdot 2^{-2n-2\mu-2}$$

quartets.

iii. Determine the key candidates involved in E_b and E_f and increase the corresponding counters. Likewise, denote the time complexity in this step as ϵ.

(c) Select the top $2^{t+m_b+m_f-m'_b-m'_f-h}$ hits in the counters to be the candidates, which delivers a h-bit or higher advantage.

(d) Guess the remaining $k - m_b - m_f$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key, where k is the key size.

Data complexity. The data complexity is $D = 2^x = (4s)^{1/4} 2^{3n/4} / \sqrt{P}$.

Memory complexity. The memory complexity is $M = D + min\{D \cdot 2^{r_b^* - \mu - 1}, D \cdot 2^{r_f^* - \mu - 1}\} + 2^{t+m_b+m_f-m'_b-m'_f} = D + min\{D^2 \cdot 2^{r_b^* - n - 1}, D^2 \cdot 2^{r_f^* - n - 1}\} + 2^{t+m_b+m_f-m'_b-m'_f}$ to store the data, the set S and key counters.

Time complexity. The time complexity of collecting data is $T_0 = D$, the time complexity of doing partial encryption and decryption under guessed key bits is

$$T_1 = 2^{m'_b+m'_f} \cdot D = 2^{m'_b+m'_f} \cdot 2^x = (4s)^{1/4} 2^{m'_b+m'_f+3n/4} / \sqrt{P},$$

the time complexity of generating set S is

$$\begin{aligned} T_2 &= 2^{m'_b+m'_f} \cdot D \cdot min\{2^{r_b^* - \mu - 1}, 2^{r_f^* - \mu - 1}\} \\ &= \sqrt{s} 2^{m'_b+m'_f+n/2} / P \cdot min\{2^{r_b - r'_b}, 2^{r_f - r'_f}\}, \end{aligned}$$

the time complexity of generating quartet candidates is

$$\begin{aligned} T_3 &= 2^{m'_b+m'_f} \cdot D^2 \cdot 2^{2r_b^*} \cdot 2^{2r_f^*} \cdot 2^{-2n-2\mu-2} \cdot \epsilon \\ &= (s \cdot 2^{m'_b+m'_f-n+2r_b+2r_f-2r'_b-2r'_f} / P^2) \cdot \epsilon \\ &= (s \cdot 2^{m'_b+m'_f+n+2r_f-2r'_b-2r'_f} / P^2) \cdot \epsilon, \end{aligned}$$

and the time complexity of exhaustive search is

$$T_4 = 2^{m'_b+m'_f} \cdot 2^{k-m'_b-m'_f-h} = 2^{k-h},$$

where $h \leq 2^{t+m_b+m_f-m'_b-m'_f}$. The overall time complexity is the sum of $T_i, i \in [0, 4]$.

A.2 Related-key Rectangle Key Recovery Framework for Ciphers with a Linear Key-Schedule

Our key recovery algorithm in the single-key setting can be easily adapted to the related-key setting for ciphers with a linear key-schedule. In related-key setting, as in [BDK05], the differential $\alpha \to \beta$ over E_0 is considered with key difference ΔK and for $\gamma \to \delta$ over E_1 the key difference is ∇K. Then the keys related to the master key K_1 are determined, where $K_2 = K_1 \oplus \Delta K$, $K_3 = K_1 \oplus \nabla K$ and $K_4 = K_1 \oplus \Delta K \oplus \nabla K$. When the key schedule is linear, the partial guess of subkeys of K_1 will determine the corresponding parts of subkeys of K_2, K_3 and K_4. Considering the related keys, the data set will be different. Choose $y = \sqrt{s} \cdot 2^{\frac{n}{2}-r_b}/P$ and we get about $s = (y \cdot 2^{2r_b})^2 \cdot 2^{-2r_b} \cdot 2^{-n} \cdot P^2$ right quartets. The related-key algorithm proceeds as follows.

1. Collect and store y structures of 2^{r_b} plaintexts each. Query the corresponding ciphertexts for each structure under the four related keys K_1, K_2, K_3 and K_4 and get the corresponding plaintext-ciphertext sets L_1, L_2, L_3 and L_4.
2. Split $(m'_b + m'_f)$-bit $k'_b \| k'_f$ into two parts: $G_L \| G_R$ where G_L has t bits.
3. Guess G_R:
   (a) Initialized a list of key counters for G_L and unguessed key bits of k_b, k_f.
   (b) Guess the t-bit G_L:
      i. For each (P_i, C_i) in data set L_i, partially encrypt P_i under the m'_b-bit $(k'_b)_i$ and partially decrypt C under the m'_f-bit $(k'_f)_i$ of $K_i (i = 1, 2, 3, 4)$. Let $P_i^* = Enc_{(k'_b)_i}(P_i)$ and $C_i^* = Dec_{(k'_f)_i}(C_i)$. For each structure under K_i $(i = 1, 2, 3, 4)$, we will get $2^{r'_b}$ sub-structures, each of which includes $2^{r_b^*}$ plaintexts. In other words, there are $y^* = y \cdot 2^{r'_b}$ structures of $2^{r_b^*}$ plaintexts each.
      ii. Let $2^{-\mu} = D \cdot 2^{-n}$. If $r_f^* - \mu \geq r_b^*$, it turns to step (A); else if $r_f^* - \mu < r_b^*$, it turns to step (D).
         A. Construct two sets as $S_1 = \{(P_1^*, C_1^*, P_2^*, C_2^*)$: $(P_1^*, C_1^*) \in L_1$, $(P_2^*, C_2^*) \in L_2$, P_1^* and P_2^* have difference in r_b^* bits$\}$, $S_2 = \{(P_3^*, C_3^*, P_4^*, C_4^*)$: $(P_3^*, C_3^*) \in L_3$, $(P_4^*, C_4^*) \in L_4$, P_3^* and P_4^* have difference in r_b^* bits$\}$. The size of both is $y \cdot 2^{r'_b} \cdot 2^{2(r_b - r'_b)} = y \cdot 2^{2r_b - r'_b} = D \cdot 2^{r_b^*}$.
         B. Insert S_1 into a hash table H_1 by $n - r_f^*$ inactive bits of C_1^* and $n - r_f^*$ inactive bits of C_2^*. Insert S_2 into a hash table H_2 by $n - r_f^*$ inactive bits of C_3^* and $n - r_f^*$ inactive bits of C_4^*.
         C. With each $2(n - r_f^*)$-bit index, we pick two distinct $(P_1^*, C_1^*, P_2^*, C_2^*)$, $(P_3^*, C_3^*, P_4^*, C_4^*)$ to generate the quartet. We will get

$$\left(\frac{|S|}{2^{2(n-r_f^*)}}\right)^2 \cdot 2^{2(n-r_f^*)} = D^2 \cdot 2^{2r_b^*} \cdot 2^{2r_f^*} \cdot 2^{-2n}$$

         quartets. Then go to step (iii).
         D. Construct two sets as $S_3 = \{(P_1^*, C_1^*, P_3^*, C_3^*)$: $(P_1^*, C_1^*) \in L_1$, $(P_3^*, C_3^*) \in L_3$, C_1^* and C_3^* are colliding in $n - r_f^*$ bits$\}$, $S_4 =$

$\{(P_2^*, C_2^*, P_4^*, C_4^*)$: $(P_2^*, C_2^*) \in L_2$, $(P_4^*, C_4^*) \in L_4$, C_2^* and C_4^* are colliding in $n - r_f^*$ bits$\}$. The size of both is $D^2 \cdot 2^{r_f - r'_f - n} = D \cdot 2^{r_f^* - \mu}$.

E. Insert S_3 into a hash table H_3 by $n - r_b^*$ inactive bits of P_1^* and $n - r_b^*$ inactive bits of P_3^*. Insert S_4 into a hash table H_4 by $n - r_b^*$ inactive bits of P_2^* and $n - r_b^*$ inactive bits of P_4^*.

F. There are at most $2^{2(n - r_b^* - \mu)}$ possible values for the $2(n - r_b^*)$-bit index. For each index, we pick two distinct entries $(P_1^*, C_1^*, P_3^*, C_3^*)$, $(P_2^*, C_2^*, P_4^*, C_4^*)$ to generate the quartet. We will get

$$\left(\frac{|S|}{2^{2(n - r_b^* - \mu)}} \right)^2 \cdot 2^{2(n - r_b^* - \mu)} = D^2 \cdot 2^{2r_b^*} \cdot 2^{2r_f^*} \cdot 2^{-2n}$$

quartets.

iii. Determine the key candidates involved in E_b and E_f and increase the corresponding counters. Likewise, denote the time complexity in this step as ϵ.

(c) Select the top $2^{t + m_b + m_f - m'_b - m'_f - h}$ hits in the counters to be the candidates, which delivers a h-bit or higher advantage.

(d) Guess the remaining $k - m_b - m_f$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key, where k is the key size.

Data complexity. The data complexity is $D_R = 4 \cdot y \cdot 2^{r_b} = 4 \cdot D$.

Memory complexity. The memory complexity is $M_R = D_R + min\{D \cdot 2^{r_b^*}, D \cdot 2^{r_f^* - \mu}\} + 2^{t + m_b + m_f - m'_b - m'_f}$ to store the data, the set S, and key counters.

Time complexity. The time complexity of collecting data is $T_0 = D_R$, the time complexity of doing partial encryption and decryption under guessed key bits is

$$T_1 = 2^{m'_b + m'_f} \cdot D_R = 2^{m'_b + m'_f} \cdot 4 \cdot y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{m'_b + m'_f + \frac{n}{2} + 2}/P,$$

the time complexity of generating set S is

$$\begin{aligned} T_2 &= 2^{m'_b + m'_f} \cdot D \cdot min\{2^{r_b^*}, 2^{r_f^* - \mu}\} \\ &= min\{\sqrt{s} \cdot 2^{m'_b + m'_f + r_b - r'_b + \frac{n}{2}}/P, s \cdot 2^{m'_b + m'_f + r_f - r'_f}/P^2\}, \end{aligned}$$

the time complexity of generating quartet candidates is

$$T_3 = 2^{m'_b + m'_f} \cdot D^2 \cdot 2^{2r_b^*} \cdot 2^{2r_f^*} \cdot 2^{-2n} \cdot \epsilon = (s \cdot 2^{m'_b + m'_f - n + 2r_b + 2r_f - 2r'_b - 2r'_f}/P^2) \cdot \epsilon,$$

and the time complexity of exhaustive search is

$$T_4 = 2^{m'_b + m'_f} \cdot 2^{k - m'_b - m'_f - h} = 2^{k-h},$$

where $h \leq 2^{t + m_b + m_f - m'_b - m'_f}$.

Adapted to the case where $r_b = n$***.*** In the related-key setting, we collect $D_R = 4 \times D = 2^{x+2}$ plaintext-ciphertext pairs, from which $(D^2)^2 \times 2^{-2n}$ quartets will be generated. Let s be the number of expected right quartets. Then, $D = (s)^{1/4} 2^{3n/4}/\sqrt{P}$ and $x = \log_2 D$.

1. Collect and store 2^x plaintexts. Query the corresponding ciphertexts for each structure under the four related keys K_1, K_2, K_3 and K_4 and get corresponding plaintext-ciphertext sets L_1, L_2, L_3 and L_4.
2. Split $(m'_b + m'_f)$-bit $k'_b \| k'_f$ into two parts: $G_L \| G_R$ where G_L has t bits.
3. Guess G_R:
   (a) Initialized a list of key counters for G_L and unguessed key bits of k_b, k_f.
   (b) Guess the t-bit G_L:
      i. For each (P_i, C_i) in data set L_i, partially encrypt P under the m'_b-bit $(k'_b)_i$ of $K_i (i = 1, 2, 3, 4)$ and partially decrypt C_i under the m'_f-bit $(k'_f)_i$ of $K_i (i = 1, 2, 3, 4)$. Let $P_i^* = Enc_{(k'_b)_i}(P_i)$ and $C_i^* = Dec_{(k'_f)_i}(C_i)$. Suppose the plaintext bits that m'_b-bit k_b corresponds to are traversed in the collected data set. Then, we will get $2^{r'_b}$ sub-sets each of $2^{x - r'_b}$ plaintexts for each related key.
      ii. Let $2^{-\mu} = D \cdot 2^{-n}$ (*i.e.*, $\mu = n - x$), $r_f^* = r_f - r'_f$ and $r_b^* = r_b - r'_b = n - r'_b$. If $r_b^* < r_f^*$, it turns to step (A); otherwise, it turns to step (D).
         A. Construct two sets as $S_1 = \{(P_1^*, C_1^*, P_2^*, C_2^*)$: $(P_1^*, C_1^*) \in L_1$, $(P_2^*, C_2^*) \in L_2$, P_1^* and P_2^* have difference in r_b^* bits$\}$, $S_2 = \{(P_3^*, C_3^*, P_4^*, C_4^*)$: $(P_3^*, C_3^*) \in L_3$, $(P_4^*, C_4^*) \in L_4$, P_3^* and P_4^* have difference in r_b^* bits$\}$. The size of both is $2^{r'_b} \cdot 2^{2(x - r'_b)} = D \cdot 2^{r_b^* - \mu}$.
         B. Insert S_1 into a hash table H_1 by $n - r_f^*$ inactive bits of C_1^* and $n - r_f^*$ inactive bits of C_2^*. Insert S_2 into a hash table H_2 by $n - (r_f - r'_f) = n - r_f^*$ inactive bits of C_3^* and $n - (r_f - r'_f) = n - r_f^*$ inactive bits of C_4^*.
         C. With each $2(n - r_f^*)$-bit index, we pick two distinct $(P_1^*, C_1^*, P_2^*, C_2^*)$, $(P_3^*, C_3^*, P_4^*, C_4^*)$ to generate the quartet. We will get

$$\left(\frac{|S|}{2^{2(n - r_f^*)}}\right)^2 \cdot 2^{2(n - r_f^*)} = D^2 \cdot 2^{2r_b^*} \cdot 2^{2r_f^*} \cdot 2^{-2n - 2\mu}$$

            quartets. Then go to step (iii).
         D. Construct two sets as $S_3 = \{ (P_1^*, C_1^*, P_3^*, C_3^*)$: $(P_1^*, C_1^*) \in L_1$, $(P_3^*, C_3^*) \in L_3$, C_1^* and C_3^* are colliding in $n - r_f^*$ bits$\}$ and $S_4 = \{ (P_2^*, C_2^*, P_4^*, C_4^*)$: $(P_2^*, C_2^*) \in L_2$, $(P_4^*, C_4^*) \in L_4$, C_2^* and C_4^* are colliding in $n - r_f^*$ bits$\}$. The size of both is $D^2 \cdot 2^{r_f - r'_f - n} = D \cdot 2^{r_f^* - \mu}$.
         E. Insert S_3 into a hash table H_3 by r'_b inactive bits of P_1^* and r'_b inactive bits of P_3^*. Insert S_4 into a hash table H_4 by r'_b inactive bits of P_2^* and r'_b inactive bits of P_4^*.

F. For each index, we pick two distinct entries $(P_1^*, C_1^*, P_3^*, C_3^*)$, $(P_2^*, C_2^*, P_4^*, C_4^*)$ to generate the quartet. We will get

$$\left(\frac{|S|}{2^{2r_b'}}\right)^2 \cdot 2^{2r_b'} = D^2 \cdot 2^{2r_b^*} \cdot 2^{2r_f^*} \cdot 2^{-2n-2\mu}$$

quartets.

iii. Determine the key candidates involved in E_b and E_f and increase the corresponding counters. Likewise, denote the time complexity in this step as ϵ.

(c) Select the top $2^{t+m_b+m_f-m_b'-m_f'-h}$ hits in the counters to be the candidates, which delivers a h-bit or higher advantage.

(d) Guess the remaining $k - m_b - m_f$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key, where k is the key size.

Data complexity. The data complexity is $D_R = 4 \cdot 2^x = 4 \cdot D = (s)^{1/4} 2^{2+3n/4}/\sqrt{P}$.

Memory complexity. The memory complexity is $M_R = D_R + min\{D \cdot 2^{r_b^*-\mu}, D \cdot 2^{r_f^*-\mu}\} + 2^{t+m_b+m_f-m_b'-m_f'}$ to store the data, the set S and key counters.

Time complexity. The time complexity of collecting data is $T_0 = D_R$, the time complexity of doing partial encryption and decryption under guessed key bits is

$$T_1 = 2^{m_b'+m_f'} \cdot D_R = 2^{m_b'+m_f'} \cdot 4 \cdot 2^x = (s)^{1/4} 2^{m_b'+m_f'+3n/4+2}/\sqrt{P}.$$

the time complexity of generating set S is

$$\begin{aligned} T_2 &= 2^{m_b'+m_f'} \cdot D \cdot min\{2^{r_b^*-\mu}, 2^{r_f^*-\mu}\} \\ &= \sqrt{s} \cdot 2^{m_b'+m_f'+\frac{n}{2}}/P \cdot min\{2^{r_b-r_b'}, 2^{r_f-r_f'}\}, \end{aligned}$$

the time complexity of generating quartet candidates is

$$\begin{aligned} T_3 &= 2^{m_b'+m_f'} \cdot D^2 \cdot 2^{2r_b^*} \cdot 2^{2r_f^*} \cdot 2^{-2n-2\mu} \cdot \epsilon \\ &= \sqrt{s} \cdot 2^{m_b'+m_f'+2r_b-2r_b'+2r_f-2r_f'-\frac{n}{2}-2\mu}/P \cdot \epsilon \end{aligned}$$

and the time complexity of exhaustive search is

$$T_4 = 2^{m_b'+m_f'} \cdot 2^{k-m_b'-m_f'-h} = 2^{k-h},$$

where $h \leq 2^{t+m_b+m_f-m_b'-m_f'}$.

A.3 Boomerang Attacks

Boomerang key recovery for $E = E_d \circ E_b$. We assume there exists a distinguisher of E_d, whose probability is P^2, input difference is α and output difference is δ. Some rounds are added before E_d.

1. Construct a set S_0 which is made up of y structures, each of 2^{r_b} plaintexts. Let $D = y \cdot 2^{r_b}$. Query and collect two sets of data:

$$S_1 = \{(P_1, C_1)|C_1 = E(P_1), P_1 \in S_0\},$$

$$S_2 = \{(P_3, C_3)|C_3 = C_1 \oplus \delta, P_3 = E^{-1}(C_3)\}.$$

2. Split m'_b-bit k'_b into two parts: $G_L \| G_R$ where G_L has t bits.
3. Guess G_R:
   (a) For each data in S_1, S_2, do partial encryptions under k'_b. Let $P_1^* = Enc_{k'_b}(P_1)$ and $P_3^* = Enc_{k'_b}(P_3)$. Then the set of obtained P_1^* contains $y \cdot 2^{r'_b}$ sub-structures.
   (b) Construct a set as

   $$S_{1,2} = \{(P_1^*, C_1, P_3^*, C_3)|C_3 = C_1 \oplus \mu, P_3^* = Enc_{m'_b}(Dec(C_3))\}.$$

   Insert $S_{1,2}$ into a hash table by $n - r_b^*$ inactive bits of P_1^* and $n - r_b^*$ inactive bits of P_3^*.
   (c) There are $y \cdot 2^{r'_b}$ possible values of the $n - r_b^*$ bits of P_1^* and $2^{n-r_b^*}$ possible values of the $n - r_b^*$ bits of P_3^*. For each index, we pick two distinct entries (P_1^*, C_1, P_4^*, C_4) and (P_3^*, C_3, P_2^*, C_2) to generate the quartet. We will get

   $$2 \cdot \binom{\frac{|S|}{2^{n-r_b^*} \cdot y \cdot 2^{r'_b}}}{2} \cdot 2^{n-r_b^*} \cdot y \cdot 2^{r'_b} = D \cdot 2^{2r_b^* - n}$$

   quartets.
   (d) Determine the key candidates involved in E_b and increase the corresponding counters. Denote the time complexity for processing one quartet as ϵ.
   (e) Select the top $2^{t+m_b-m'_b-h}$ hits in the counters to be the candidates, which delivers a h-bit or higher advantage.
   (f) Guess the remaining $k - m_b$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key, where k is the key size.

Data complexity. The data complexity is $D_B = 2D = 2 \cdot y \cdot 2^{r_b} = 4s/P^2$.

Memory complexity. The memory complexity of this framework is $M = D_B + D + 2^{t+m_b-m'_b}$ to store the data, the set $S_{1,2}$ and key counters.

Time complexity. The time complexity of collecting data is $T_0 = D_B$, the time complexity of doing partial encryption and decryption under guessed key bits is

$$T_1 = 2^{m'_b} \cdot D_B = 2^{m'_b} \cdot 2 \cdot y \cdot 2^{r_b} = s \cdot 2^{m'_b+2}/P^2,$$

the time complexity of generating set $S_{1,2}$ is

$$T_2 = 2^{m'_b} \cdot D = s \cdot 2^{m'_b+1}/P^2,$$

the time complexity of generating quartet candidates is

$$T_3 = 2^{m_b'} \cdot D \cdot 2^{2r_b^*} \cdot 2^{-n} \cdot \epsilon = s \cdot 2^{m_b'+2r_b-2r_b'-n+1}/P^2,$$

and the time complexity of exhaustive search is

$$T_4 = 2^{m_b'} \cdot 2^{k-m_b'-h} = 2^{k-h},$$

where $h \leq 2^{t+m_b-m_b'}$.

Related-key boomerang key recovery for $E = E_f \circ E_d$**.** We assume there exists a distinguisher of E_d, whose probability is P^2, input difference is α and output difference is δ. Some rounds are added after E_d. In related-key setting, the differential $\alpha \to \beta$ over E_0 is considered with key difference ΔK and for $\gamma \to \delta$ over E_1 the key difference is ∇K. Then the keys related to the master key K_1 are determined, where $K_2 = K_1 \oplus \Delta K$, $K_3 = K_1 \oplus \nabla K$ and $K_4 = K_1 \oplus \Delta K \oplus \nabla K$. When the key schedule is linear, the partial guess of subkeys of K_1 will determine the corresponding parts of subkeys of K_2, K_3 and K_4. The related-key algorithm proceeds as follows.

1. Construct a set S_0 which is made up of y structures, each of 2^{r_f} ciphertexts. Query the corresponding ciphertexts for each structure under the four related keys K_1, K_2, K_3 and K_4 and get the corresponding plaintext-ciphertext sets S_1, S_2, S_3 and S_4. Let $D = y \cdot 2^{r_f}$. Query and collect four sets of data:

$$S_1 = \{(P_1, C_1)|P_1 = E_{k^1}^{-1}(C_1), C_1 \in S_0\},$$

$$S_2 = \{(P_2, C_2)|P_2 = P_1 \oplus \alpha, C_2 = E_{k^2}(P_2), P_1 \in S_1\},$$

$$S_3 = \{(P_3, C_3)|P_3 = E_{k^3}^{-1}(C_3), C_3 \in S_0\},$$

$$S_4 = \{(P_4, C_4)|P_4 = P_3 \oplus \alpha, C_4 = E_{k^4}(P_4), P_3 \in S_3\}.$$

2. Split m_f'-bit k_f' into two parts: $G_L \| G_R$ where G_L has t bits.
3. Guess G_R:
   (a) Initialized a list of key counters for G_L and unguessed key bits of k_f.
   (b) Guess the t-bit G_L:
      i. For each data in S_1, S_2, S_3, S_4, do partial decryptions under $(k_f')^i$. Let $C_i^* = Dec_{(k_f')^i}(C_i)$. Then the set of obtained C_1^* contains $y \cdot 2^{r_f'}$ sub-structures, each of $2^{r_f^*}$ ciphetexts.
      ii. Construct a set as

$$S_{1,2} = \{(P_1, C_1^*, P_2, C_2^*)|P_2 = P_1 \oplus \alpha, C_2^* = Dec_{(k_f')^1}(Enc(P_2))\},$$

$$S_{3,4} = \{(P_3, C_3^*, P_4, C_4^*)|P_4 = P_3 \oplus \alpha, C_4^* = Dec_{(k_f')^2}(Enc(P_4))\},$$

Insert $S_{1,2}, S_{3,4}$ into a hash table by $n - r_f^*$ inactive bits of C_1^*, C_3^* and $n - r_f^*$ inactive bits of C_2^*, C_4^*.

iii. There are $y \cdot 2^{r'_f}$ possible values for the $n - r^*_f$ bits of C^*_1 and $2^{n-r^*_f}$ possible values for the $n - r^*_f$ bits of C^*_2. For each index, we pick two distinct entries (P_1, C^*_1, P_2, C^*_2) and (P_3, C^*_3, P_4, C^*_4) to generate the quartet. The number of quartet we will get is

$$(\frac{|S_{1,2}|}{2^{n-r^*_f} \cdot y \cdot 2^{r'_f}})^2 \cdot 2^{n-r^*_f} \cdot y \cdot 2^{r'_f} = D \cdot 2^{2r^*_f - n}.$$

iv. Determine the key candidates involved in E_f and increase the corresponding counters. Denote the time complexity for processing one quartet as ϵ.

(c) Select the top $2^{t+m_f-m'_f-h}$ hits in the counters to be the candidates, which delivers a h-bit or higher advantage.

(d) Guess the remaining $k - m_f$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key, where k is the key size.

Data complexity. From y structures, we can form $y \cdot 2^{2r_f}$ plaintext pairs. Among them, $y \cdot 2^{r_f}$ pairs satisfy δ difference on average. Let s be the expected number of right quartets, so we have $y \cdot 2^{r_f} \cdot P^2 = s$, $y = s \cdot 2^{-r_f}/P^2$ and $D = y \cdot 2^{r_f} = s/P^2$. Therefore, the data complexity is $D_{RB} = 4D = 4s/P^2$.

Memory complexity. The memory complexity of is $M_{RB} = D_{RB} + D + 2^{t+m_f-m'_f}$ to store the data, the set $S_{1,2}$ and the counters.

Time complexity. The time complexity of collecting data is $T_0 = D_{RB}$, the time complexity of doing partial encryption and decryption under guessed key bits is

$$T_1 = 2^{m'_f} \cdot D_{RB} = 2^{m'_f} \cdot 4 \cdot y \cdot 2^{r_f} = s \cdot 2^{m'_f+2}/P^2,$$

the time complexity of generating set S is

$$T_2 = 2^{m'_f} \cdot D = s \cdot 2^{m'_f}/P^2,$$

the time complexity of generating and processing quartet candidates is

$$T_3 = 2^{m'_f} \cdot D \cdot 2^{2r^*_f} \cdot 2^{-n} \cdot \epsilon = s \cdot 2^{m'_f+2r_f-2r'_f-n}/P^2 \cdot \epsilon,$$

and the time complexity of exhaustive search is $T_4 = 2^{m'_f} \cdot 2^{k-m'_f-h} = 2^{k-h}$, where $h \leq 2^{t+m_f-m'_f}$.

B Application to Some Other Ciphers

B.1 Other Variants of SKINNY

Attack on 32-round SKINNY-128-384. We reuse the 23-round rectangle distinguisher of SKINNY-128-384 from [DQSW22]. The probability of this distinguisher is $2^{-n}P^2 = 2^{-128} \cdot 2^{-115.09}$. Our key recovery extends the distinguisher by four rounds at the top and five rounds at the bottom, as shown in Figure 9. The parameters for this attack are: $r_b = 12 \times 8$, $r_f = 16 \times 8$, $m_b = 18 \times 8$ and $m_f = 24 \times 8$. Note that $k_b \cup k_f$ only contain $(18 + 24 - 2) \times 8$ information bits.

We apply our generic framework and obtain that when constructing pairs on the top and guessing 27 subtweakey cells leads to lowest complexity overall. The positions of the guessed subtweakey cell and 19 filters ($r'_b = 11 \times 8, r'_f = 8 \times 8$) that can be checked under these subtweakey cells are mark by red squares in Figure 9.

Next, we compute the complexities of our attack.

- The data complexity is $D_R = 4 \cdot y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{n/2+2}/P = \sqrt{s} \cdot 2^{123.54}$.
- The memory complexity is $M_R = D_R + D \cdot 2^{r_b^*} + 2^{t+m_b+m_f-m'_b-m'_f} = \sqrt{s} \cdot 2^{123.54} + \sqrt{s} \cdot 2^{129.54} + 2^{104+t}$
- The time complexity $T_1 = 2^{m'_b+m'_f} \cdot D_R = \sqrt{s} \cdot 2^{27\times8+123.55} = \sqrt{s} \cdot 2^{339.54}$;
- $T_2 = 2^{m'_b+m'_f} \cdot D \cdot 2^{r_b-r'_b} = \sqrt{s} \cdot 2^{27\times8+121.55+8} = \sqrt{s} \cdot 2^{345.54}$;
- $T_3 = 2^{m'_b+m'_f} \cdot D^2 \cdot 2^{2r_b^*+2r_f^*-2n} \cdot \epsilon = s \cdot 2^{27\times8+243.09+2\times8+2\times64-2\times128} \cdot \epsilon = s \cdot 2^{347.09} \cdot \epsilon$;
- $T_4 = 2^{384-h}, h < 104 + t$.

Processing a candidate quartet to retrieve the rest of k_b and k_b can be realized by looking up tables. We pre-compute several tables as illustrated in Table 10, so that ϵ is equivalent to about 4 memory accesses which is around $4 \times \frac{1}{16} \times \frac{1}{32} = 2^{-7}$ encryption. If we set $s = 1, h = 40$ and $t = 0$, then the data, memory and time complexities of our attack are $2^{123.54}$, $2^{129.54}$, and $2^{344.16}$, respectively. The success probability is about 82.1%.

The comparison with the previous rectangle attacks based on the same distinguisher is presented in Table 11.

Attack on 26-round `SKINNY-128-256`. Applying our new rectangle key recovery algorithm to `SKINNY-128-256`, we get a new 26 rectangle attack by appending 3-round E_b and 4-round E_f, with using the 19-round rectangle distinguisher of `SKINNY-128-256` in [DQSW22], as shown in Figure 10. The probability of the distinguisher is $2^{-n}P^2 = 2^{-128-121.07} = 2^{249.07}$. The parameters for this attack are: $r_b = 9 \times 8 = 72, r_f = 12 \times 8 = 96, m_b = 11 \times 8 = 88$ and $m_f = 21 \times 8 = 168$. Due to the tweakey schedule, $k_b \cup k_f$ only contain $(88+168-16) = 240$ information bits.

The best guessing parameters are $m'_b = 72, r'_b = 64$ and $m'_f = r'_f = 32$, which means guessing partial bits of k_b and k_f. This type of guessing is not covered in previous rectangle key recovery attacks. The complexities of our new attack are as follow.

- The data complexity is $D_R = 4 \cdot y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{n/2+2}/P = \sqrt{s} \cdot 2^{126.53}$.
- The memory complexity is $M_R = D_R + D \cdot 2^{r_b^*} + 2^{t+m_b+m_f-m'_b-m'_f} = \sqrt{s} \cdot 2^{126.53} + \sqrt{s} \cdot 2^{132.53} + 2^{136+t}$.
- The time complexity $T_1 = 2^{m'_b+m'_f} \cdot D_R = \sqrt{s} \cdot 2^{13\times8+126.53} = \sqrt{s} \cdot 2^{230.53}$;

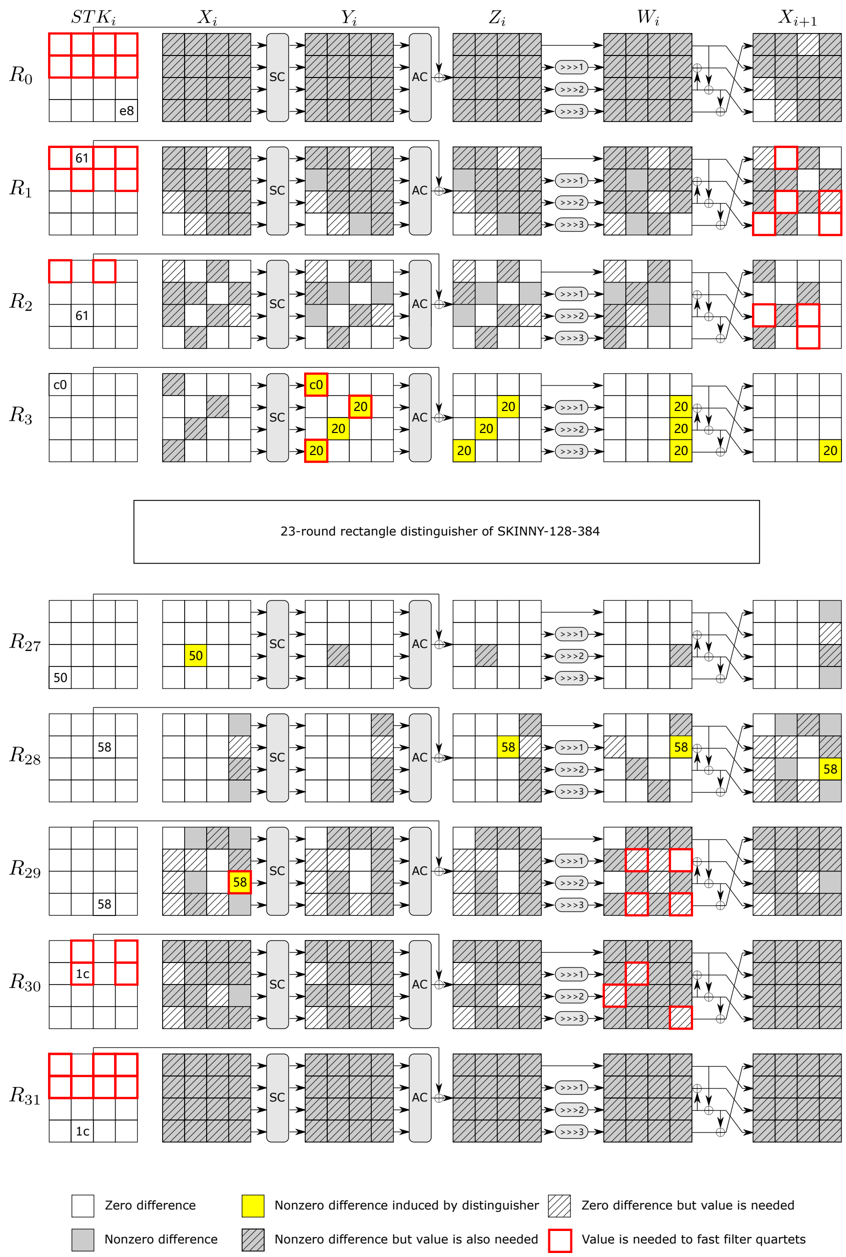

Figure 9: A 32-round key recovery attack against SKINNY-128-384

– $T_2 = 2^{m'_b+m'_f} \cdot D \cdot 2^{r_b-r'_b} = \sqrt{s} \cdot 2^{13\times 8+124.53+8} = \sqrt{s} \cdot 2^{236.53}$;

– $T_3 = 2^{m'_b+m'_f} \cdot D^2 \cdot 2^{2r^*_b+2r^*_f-2n} \cdot \epsilon = s \cdot 2^{13\times 8+124.53\times 2+2\times 8+2\times 64-2\times 128} \cdot \epsilon = s \cdot 2^{241.07} \cdot \epsilon$;

Table 10: Precomputation tables for 32-round attack on SKINNY-128-384, where underlined bytes are used as input and determine the time and memory complexity for building the table. Note that the precomputation table may built for pairs or quartets. When a table is built for pairs, the filter effect in brackets is for two pairs.

No.	Starting cells	subtweakey bytes	Bytes deduced	Filter	Pairs or quartets	Time and memory	filter effect
1	$\underline{Z_{31}[1]}$, $\underline{Z_{30}[6]}$, $\underline{X_{31}[13]}$, $\underline{X'_{31}[13]}$	$\underline{STK_{31}[1]}$, $\underline{STK_{30}[6]}$	$X_{30}[14], X_{30}[6], Z'_{31}[1], Z'_{30}[6]$	$X_{30}[14] \oplus X_{30}[6] \oplus X'_{30}[14] \oplus X'_{30}[6] = 0$	Quartets	2^{96}	1
	$(Z^i_{31}[1], X^i_{31}[13], Z^i_{30}[6]), i = 1, 2, 3, 4 : STK_{31}[1], STK_{30}[6]$						
2	$\underline{Z_{30}[0]}$, $\underline{X_{30}[12]}$, $\underline{Z_{29}[1]}$, $\underline{\Delta X_{29}[9]}$	$\underline{STK_{30}[0]}$, $\underline{STK_{29}[1]}$	$X_{29}[13], X_{29}[1], Z'_{30}[0], Z'_{29}[1]$	$X_{29}[13] \oplus X'_{29}[13] = \Delta X_{29}[9]$, $X_{29}[1] \oplus X_{29}[13] \oplus X'_{29}[1] \oplus X'_{29}[13] = 0$	Quartets	2^{80}	2^{-16}
	$(Z^i_{30}[0], Z^i_{29}[1], X^j_{30}[12], \Delta X^j_{29}[9]), i = 1, 2, 3, 4, j = 1, 3 : STK_{30}[0], STK_{29}[1]$						
3	$\underline{Z_{30}[2]}$, $\underline{Z_{30}[4]}$, $\underline{X_{30}[14]}$, $\underline{X'_{30}[14]}$, $\underline{X_{30}[8] \oplus X_{30}[12]}$, $\underline{X'_{30}[8] \oplus X'_{30}[12]}$, $\underline{Z_{29}[3]}$	$\underline{STK_{30}[2]}$, $\underline{STK_{30}[4]}$, $\underline{STK_{29}[3]}$, $\underline{STK_{29}[7]}$	$X_{29}[3], X_{29}[7], X_{29}[15], Z'_{30}[2], Z'_{30}[4], Z'_{29}[3]$	$X_{29}[3] \oplus X_{29}[15] = X'_{29}[3] \oplus X'_{29}[15]$, $X_{29}[7] \oplus X_{29}[15] = X'_{29}[7] \oplus \underline{X'_{29}[15]}$	Pairs	2^{96}	2^{16} (1)
	$(Z^i_{30}[2], Z^i_{30}[4], X^i_{30}[14], X^i_{30}[8] \oplus X^i_{30}[12], Z^i_{29}[3]), i = 1, 2 : STK_{30}[2], STK_{30}[4], STK_{29}[3], STK_{29}[7]$						
4	$\underline{Z_{29}[2]}$, $\underline{Z_{29}[5]}$, $\underline{X_{29}[13]}$, $\underline{X_{29}[14]}$, $\underline{\Delta X_{28}[3]}$	$\underline{STK_{29}[2]}$, $\underline{STK_{29}[5]}$	$X_{28}[11], X_{28}[15], Z'_{29}[2], X'_{29}[13]$	$X_{28}[15] \oplus X'_{28}[15] = \Delta X_{28}[3]$, $X_{28}[15] \oplus X'_{28}[15] = X_{28}[11] \oplus X'_{28}[11]$	Quartets	2^{96}	2^{-16}
	$(Z^i_{29}[2], X^i_{29}[13], Z^j_{29}[5], X^j_{29}[14], \Delta X^j_{28}[3]), i = 1, 2, 3, 4, j = 1, 3 : STK_{29}[2], STK_{29}[5]$						
5	$\underline{Z_{29}[4]}$, $\underline{X_{29}[8] \oplus X_{29}[12]}$, $\underline{X_{28}[15]}$, $\underline{STK_{28}[7]}$	$\underline{STK_{29}[4]}$	$X_{27}[9], X'_{28}[15]$	$X_{27}[9] \oplus X'_{27}[9] = 0x50$	Quartets	2^{64}	2^{-8}
	$(X^i_{28}[15], Z^j_{29}[4], X^j_{29}[8] \oplus X^j_{29}[12], STK_{28}[7]), i = 1, 2, 3, 4, j = 1, 3 : STK_{29}[4]$						
6	$\underline{Y_1[6]}$, $\underline{Y_1[9]}$, $\underline{Y_2[4]}$	$\underline{STK_1[6]}$, $\underline{STK_2[4]}$	$W_2[5], W_2[9], Y'_2[4]$	$Y_3[9] \oplus Y'_3[9] = 0x20$	Quartets	2^{64}	1
	$(Y^i_2[4], Y^j_1[6], Y^j_1[9]), i = 1, 2, 3, 4, j = 1, 3 : STK_1[6], STK_2[4]$						

– $T_4 = 2^{256-h}, h < 136 + t$.

Processing a candidate quartet to retrieve the rest of k_b and k_f can be realized by looking up tables. We pre-compute several tables as illustrated in Table 12, so that ϵ is equivalent to about $1 + 1 + 2^8 + 2^8 + 1 + 1 = 516$ memory accesses which is around $516 \times \frac{1}{16} \times \frac{1}{26} = 2^{0.31}$ encryption. If we set $s = 1, h = 40$ and $t = 0$, then the data, memory and time complexities of our attack are $2^{126.53}$, 2^{136}, $2^{241.38}$, respectively. The success probability is about 64.06%.

The comparison with the previous rectangle attacks based on the same distinguisher is presented in Table 13.

Table 11: Comparisons of key recovery attacks on SKINNY-128-384

P^2	m_b, m_f	m'_b, m'_f	Data	Memory	Time	Reference
$2^{-115.09}$	144, 192	144, 88	$2^{123.54}$	$2^{123.54}$	$2^{354.99}$	[HBS21]
$2^{-115.09}$	144, 192	128, 88	$2^{123.54}$	$2^{129.54}$	$2^{344.16}$	This

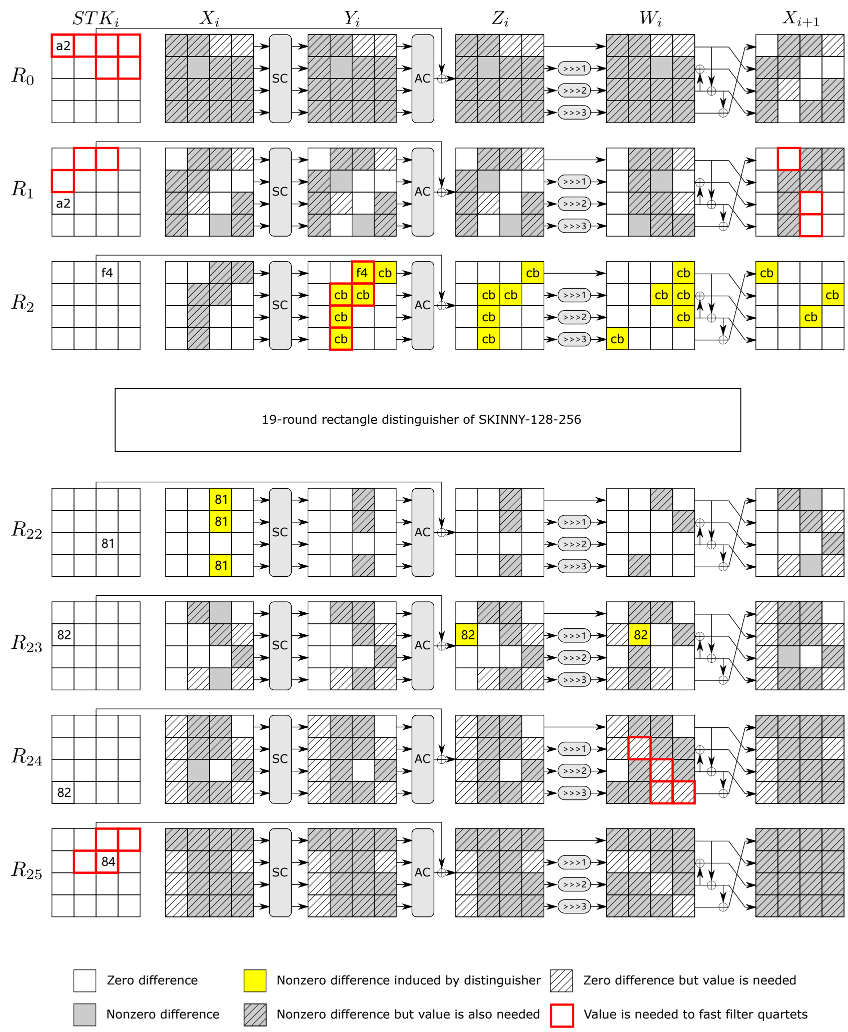

Figure 10: A 26-round key recovery attack against SKINNY-128-256

B.2 Deoxys-BC-256

Deoxys-BC [JNPS16] is the internal tweakable block cipher of Deoxys-II, which is among the final portfolio of CAESAR competition.Both versions of the cipher

Table 12: Precomputation tables for 26-round attack on `SKINNY`-128-256, where underlined bytes are used as input and determine the time and memory complexity for building the table. Note that the precomputation table may built for pairs or quartets. When a table is built for pairs, the filter effect in brackets is for two pairs.

No.	Starting cells	Subtweakey bytes	Bytes deduced	Filter	Pairs or quartets	Time and memory	Filter effect
1	$\underline{Z_{25}[1]}, \underline{X_{25}[6]}$ $\underline{X_{25}[13]}, \underline{X'_{25}[13]}$	$\underline{STK_{25}[1]}$, $\underline{STK_{24}[2]}$	$X_{24}[2], X_{24}[14], Z'_{25}[1], X'_{25}[6]$	$X_{24}[2] \oplus X'_{24}[2] = X_{24}[14] \oplus \underline{X'_{24}[14]}$	Quartets	2^{96}	1
	$(Z^i_{24}[1], X^i_{24}[13], X^i_{24}[6]), i = 1, 2, 3, 4 : STK_{24}[1], STK_{23}[2]$						
2	$\underline{Z_{25}[7]}$, $\underline{X_{25}[11] \oplus X_{25}[15]}$, $\underline{\Delta X_{24}[14]}$	$\underline{STK_{25}[7]}$, $\underline{STK_{24}[6]}$	$X_{24}[6], Z'_{25}[7], X'_{25}[11] \oplus X'_{25}[15]$	$X_{24}[6] \oplus X'_{24}[6] = \Delta X_{24}[14]$	Quartets	2^{80}	1
	$(Z^i_{25}[7], X^i_{25}[11] \oplus X^i_{25}[15], \Delta X^j_{24}[14]), i = 1, 2, 3, 4, j = 1, 3 :$ $STK_{25}[7], STK_{24}[6]$						
3	$\underline{Z_{24}[5]}, \underline{X_{24}[9]}$, $\underline{Z_{25}[0]}, \underline{X_{25}[12]}$, $\underline{X'_{24}[9]}, \underline{X'_{25}[12]}$	$\underline{STK_{25}[0]}$, $\underline{STK_{24}[5]}$	$X_{24}[5], X_{24}[12]$ $Z'_{24}[5], Z'_{25}[0]$,	$X_{24}[5] \oplus X'_{24}[5] \oplus X_{24}[9] \oplus X'_{24}[9] \oplus X_{24}[13] \oplus \underline{X'_{24}[13]} = 0x82$	Pairs	2^{72}	2^8 (1)
	$(Z^i_{25}[0], X^i_{25}[12], Z^i_{24}[5], X^i_{24}[9]), i = 1, 2 : STK_{25}[0], STK_{24}[5]$						
4	$\underline{Z_{23}[2]}, \underline{X_{24}[13]}$, $\underline{Z_{24}[1]}, \underline{X'_{24}[13]}$,	$\underline{STK_{24}[1]}$, $\underline{STK_{23}[2]}$	$X_{23}[2], Z'_{23}[2], Z'_{24}[1]$	$X_{23}[2] \oplus X'_{23}[2] = \underline{\Delta X_{23}[14]}$	Quartets	2^{96}	1
	$(Z^i_{24}[1], Z^i_{23}[2], X^i_{24}[13]), i = 1, 2, 3, 4 : STK_{24}[1], STK_{23}[2]$						
5	$\underline{Z_{24}[7]}, \underline{\Delta X_{23}[14]}$, $\underline{X_{24}[11] \oplus X_{24}[15]}$, $\underline{STK_{22}[2]}$	$\underline{STK_{24}[7]}$, $\underline{STK_{23}[6]}$	$X_{23}[6], X_{22}[2]$ $X'_{24}[11] \oplus X'_{24}[15]$	$X_{23}[6] \oplus X'_{23}[6] = \Delta X_{23}[14]$, $X_{22}[2] \oplus X_{22}[2] = 0x81$	Quartets	2^{56}	2^{-16}
	$(X^i_{24}[11] \oplus X^i_{24}[15], Z^j_{24}[7], \Delta X^j_{23}[14], STK_{22}[6]), i = 1, 2, 3, 4,$ $j = 1, 3 : STK_{24}[7], STK_{23}[6]$						
6	$\underline{Z_{25}[4]}, \underline{Z_{24}[0]}$, $\underline{Z_{23}[1]}, \underline{X_{24}[12]}$ $\underline{STK_{23}[1]}$	$\underline{STK_{25}[4]}$, $\underline{STK_{24}[0]}$	$X_{22}[14], Z'_{23}[1]$	$X_{22}[14] \oplus X'_{22}[14] = 0x81$	Quartets	2^{88}	1
	$(Z^i_{23}[1], Z^j_{25}[4], Z^j_{24}[0], X^j_{24}[12], STK_{23}[1]), i = 1, 2, 3, 4, j = 1, 3 :$ $STK_{25}[4], STK_{24}[0]$						
7	$\underline{Z_{24}[4]}$, $\underline{X_{23}[11] \oplus X_{23}[15]}$, $\underline{X_{24}[13]}, \underline{STK_{22}[6]}$	$\underline{STK_{24}[4]}$ $\underline{STK_{23}[7]}$	$X_{22}[6]$, $X'_{23}[11] \oplus X'_{23}[15]$,	$X_{22}[6] \oplus X'_{22}[6] = 0x81$	Quartets	2^{80}	1
	$(X^i_{23}[11] \oplus X^i_{23}[15], X^j_{24}[13], Z^j_{24}[4], STK_{23}[2], STK_{22}[6])$, $i = 1, 2, 3, 4 : STK_{24}[7], STK_{23}[6]$						
8	$\underline{Y_0[4]}, \underline{W_1[15]}$ $\underline{Y_1[3]}, Y'_0[4]$	$\underline{STK_0[4]}$, $\underline{STK_1[3]}$	$Y_2[9]$, $W_1[9]$,	$Y_2[3] \oplus Y'_2[3] = 0xcb$	Quartets	2^{80}	1
	$(Y^i_0[4], W^i_1[15], Y^j_1[3]), i = 1, 2, 3, 4, j = 1, 3 : STK_0[4], STK_1[3]$						

have 128-bit state and variable size key and tweak. It has two versions with a 256-bit key size and 384-bit key size.

Rectangle Attack. We reuse the 9-round rectangle distinguisher of `Deoxys-BC`-256 proposed by Cid *et al.* [CHP+17, WP19] to attack 11-round rectangle Deoxys-BC-256 with 2-round E_f, as shown in Figure 7. The probability of the distinguisher

Table 13: Comparisons of key recovery attacks on SKINNY-128-256

P^2	m_b, m_f	m'_b, m'_f	Data	Memory	Time	Reference
$2^{-121.07}$	$88, 168$	$88, 24$	$2^{126.53}$	2^{136}	$2^{254.4}$	[HBS21]
$2^{-121.07}$	$88, 168$	$72, 32$	$2^{126.53}$	2^{136}	$2^{241.38}$	This

is $2^{-n}P^2 = 2^{-128-120.4} = 2^{-248.4}$, and other parameteres are: $n = 128, k = 256, m_b = r_b = 0, m_f = (16 + 10) \times 8 = 208, r_f = 16 \times 8 = 128$.

The best guessing parameters are $m'_f = 12 \times 8 = 96$ and $r'_f = 8 \times 8 = 64$, which means guessing 8 bytes of k_f. The complexities of our new attack are as follow.

- The data complexity is $D_R = 4 \cdot y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{n/2+2}/P = \sqrt{s} \cdot 2^{126.2}$.
- The memory complexity is $M_R = D_R + D \cdot 2^{r_b^*} + 2^{t+m_b+m_f-m'_b-m'_f} = \sqrt{s} \cdot 2^{126.2} + \sqrt{s} \cdot 2^{124.2} + 2^{112+t}$.
- The time complexity $T_1 = 2^{m'_b+m'_f} \cdot D_R = \sqrt{s} \cdot 2^{12\times 8+126.2} = \sqrt{s} \cdot 2^{222.2}$;
- $T_2 = 2^{m'_b+m'_f} \cdot D \cdot 2^{r_b-r'_b} = \sqrt{s} \cdot 2^{12\times 8+124.2} = \sqrt{s} \cdot 2^{220.2}$;
- $T_3 = 2^{m'_b+m'_f} \cdot D^2 \cdot 2^{2r_b^*+2r_f^*-2n} \cdot \epsilon = s \cdot 2^{12\times 8+124.2\times 2+2\times 64-2\times 128} \cdot \epsilon = s \cdot 2^{216.4} \cdot \epsilon$;
- $T_4 = 2^{256-h}, h < 112 + t$.

Processing a candidate quartet to retrieve the rest of k_f and can be realized by looking up tables. We consider the equivalent round subtweakey $MTK_i = SR^{-1} \circ MC^{-1}(STK_i)$ in round i. To process a candidate quartet to retrieve the rest of k_f,we prepare some tables as illustrated in Table 8. So that ϵ is equivalent to about 1 memory access which is around $1 \times \frac{1}{16} \times \frac{1}{11} = 2^{-7.45}$ encryption. If we set $s = 1.5, h = 36$ and $t = 0$, then the data, memory and time complexities of our attack are $2^{126.78}$, 2^{128}, $2^{222.49}$, respectively. The success probability is about 77.19%.

Table 14: Comparisons of key recovery attacks on Deoxys-BC-256

P^2	m_b, m_f	m'_b, m'_f	Data	Memory	Time	Reference
2^{-122}	0,80	0,0	$2^{127.58}$	$2^{127.58}$	2^{204}	[CHP+17]
$2^{-116.4}$	128,80	112,0	$2^{122.1}$	$2^{128.2}$	$2^{249.9}$	[ZDJ19a]
$2^{-120.4}$	0,208	0,96	$2^{126.78}$	2^{128}	$2^{222.49}$	This

C Previously Proposed Key Recovery Algorithms

C.1 Algorithm 1: Biham-Dunkelman-Keller's Algorithm at EUROCRYPT 2001

Biham, Dunkelman and Keller introduced the rectangle attack [BDK01] at EUROCRYPT 2001 and first applied it to Serpent [ABK98]. The specific procedures are as follows:

1. Create and store $y = \sqrt{s} \cdot 2^{\frac{n}{2}-r_b+1}/P$ structures including 2^{r_b} each by tranversing the active bits in each structure, where s denotes the expected number of right quartets.
2. Initialize $2^{m_b+m_f}$ key counters for the $(m_f + m_b)$-bit subkey involved in E_b and E_f. For each $(m_f + m_b)$-bit subkey and each structure:
   (a) Partially encrypt plaintext P_1 to the position of α under the guessed m_b-bit subkey in E_b and partially decrypt the state xored the known difference α to the plaintext P_2.
   (b) Denote C_1 and C_2 the corresponding ciphertexts of P_1 and P_2 respectively. Partially decrypt C_1 to the position of δ and encrypt it to the ciphertext C_3 after xoring δ. Similarly, we find C_4 from C_2 in the same way and then obtain the quartet (C_1, C_2, C_3, C_4).
   (c) Check whether the corresponding ciphertexts (C_3, C_4) exist in our data. If exist, we check the difference is α after partially encrypting corresponding plaintexts (P_3, P_4) under m_b-bit subkey in E_b. If so, we increase the corresponding counter by 1.
   (d) Select the top $2^{m_b+m_f-h}$ hits in the counter to be the candidates, which delivers a h-bit or higher advantage.
   (e) Guess the remaining $k - m_b - m_f$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key, where k is the key size.

The data complexity is $D = y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{\frac{n}{2}+1}/P$. The memory complexity is $D + 2^{m_b+m_f}$ to store the data and key counters. The time complexity of generating quartets and determine the key candidates is

$$T_1 = 2^{m_b+m_f} \cdot D = \sqrt{s} \cdot 2^{m_b+m_f+\frac{n}{2}+1}/P,$$

and the complexity of exhaustive search is

$$T_2 = 2^{m_b+m_f-h} \cdot 2^{k-(m_b+m_f)} = 2^{k-h},$$

where $h \leq m_b + m_f$.

C.2 Algorithm 2: Biham-Dunkelman-Keller's Algorithm at FSE 2002

At FSE 2002, Biham, Dunkelman and Keller further introduced their new algorithm for using rectangle distinguisher in key recovery attack in single-key setting. Later, the attack was converted into related-key setting by Liu *et al.* [LGS17] on ciphers with linear key schedule. The procedures are summarized as follows and more details are described in [BDK01]:

1. Construct and store y structures of 2^{r_b} plaintexts each by tranversing the active bits in each structure.
2. Initialize an array of $2^{m_b+m_f}$ counters for the $(m_f + m_b)$-bit subkey involved in E_b and E_f.
3. Insert the $y \cdot 2^{r_b}$ ciphertexts into a hash table H according to the $n - r_f$ inactive ciphertext bits. For each index, there are $2^{r_b} \cdot 2^{r_f - n}$ plaintexts and ciphertexts colliding in the $n - r_f$ bits for each structure.
4. In each structure S, we search for a ciphertext pair (C_1, C_2) and choose a ciphertext C_3 by the $n - r_f$ inactive ciphertext bits of C_1 from the hash table H. We pick a ciphertext C_4 according to the $n - r_f$ inactive ciphertext bits of C_2 from the hash table in the same way. Then we check whether the corresponding plaintexts P_3 and P_4 are in the same structure. If so, then we generate a quartet (P_1, P_2, P_3, P_4) and its corresponding ciphertexts (C_1, C_2, C_3, C_4).
5. Determine the key candidates involved in E_b and E_f with the quartets obtained above and increase the corresponding counters. This phase is just a guess and filter procedure. Denote the time complexity in this step as ϵ_2.
6. Select the top $2^{m_b+m_f-h}$ hits in the counter to be the candidates, which delivers a h-bit or higher advantage.
7. Guess the remaining $k - m_b - m_f$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key, where k is the key size.

The data complexity is $D = y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{\frac{n}{2}+1}/P$. The memory complexity is $D + 2^{m_b+m_f}$ to store the data and key counters. The time complexity of inserting the ciphertexts into the hash table is

$$T_1 = D = \sqrt{s} \cdot 2^{\frac{n}{2}+1}/P,$$

the time complexity to generate quartets accessing the colliding pairs is

$$T_2 = \binom{D}{2} \cdot 2^{r_f-n} = D^2 \cdot 2^{r_f-n-1} = y^2 \cdot 2^{2r_b+r_f-n-1} = s \cdot 2^{r_f+1}/P^2,$$

the complexity of determining the key candidates is

$$\begin{aligned} T_3 &= (y \cdot 2^{2r_b+r_f-n-1})^2 = y^2 \cdot 2^{4r_b+2r_f-2n-2} \cdot \epsilon_2 \\ &= D^2 \cdot 2^{2r_b+2r_f-2n-2} \cdot \epsilon_2 \\ &= s \cdot 2^{2r_b+2r_f-n}/P^2 \cdot \epsilon_2 \end{aligned}$$

and the complexity of exhaustive search is

$$T_4 = 2^{m_b+m_f-h} \cdot 2^{k-(m_b+m_f)} = 2^{k-h},$$

where $h \leq m_b + m_f$.

In step 4 of the above algorithm, quartets are constructed in time T_2 and the memory cost does not exceed D. Here we try to give an illustration of how to avoid the increase of the memory. Firstly, we need to store the collected data and the memory complexity is D. Next, insert (P, C) into a hash table H_1 according to $n - r_f$ bits in ciphertexts. There are $D \cdot 2^{-(n-r_f)}$ values in each index and the time complexity of this step is D. Then for each structure $S_i(i = 1, 2, \cdots, y)$, considering $(P_1, P_2)(P_1, P_2 \in S_i)$, we will obtain 2^{2r_b-1} such pairs. The values in the same index with C_1 in H_1 are denoted as $C_3^1, C_3^2, \cdots, C_3^j, \cdots$ and $C_4^1, C_4^2, \cdots, C_4^k, \cdots$ for C_2. Insert C_3^j into a hash table H_2 according to $n - r_b$ bits of P_3^j, which is the same as we defined in our algorithm. We then look up H_2 with $n - r_b$ bits of P_4^k; if a collision is found, (C_1, C_2, C_3^j, C_4^k) is a candidate quartet. In this step, the memory complexity of storing H_2 is $D \cdot 2^{-(n-r_f)}$, which can be ignored compared to D. We will get

$$y \cdot 2^{2r_b-1} \cdot \binom{D \cdot 2^{-(n-r_f)}}{2} \cdot 2^{-(n-r_b-\mu)} = D^2 \cdot 2^{2r_b+2r_f-2n-2}$$

candidate quartets.However, there will be an extra time complexity of accessing the hash table H_2, which is

$$y \cdot 2^{2r_b-1} \cdot D \cdot 2^{-(n-r_f)} = D^2 \cdot 2^{r_b+r_f-n-1}.$$

It should be noted that the extra time complexity may not be omitted as it may be a dominant part in some cases. We feel that this term of time complexity was neglected by the authors of [BDK01] inadvertently, or the memory complexity should be higher than D.

C.3 Algorithm 3: Zhao *et al.*'s Single-key Variant

Zhao et al. proposed a new generalized related-key rectangle framework [ZDJ19b, ZDM+20] for block ciphers with linear key schedule. The attack can be applied to single-key setting with simple modifications:

1. Collect and store y structures of 2^{r_b} plaintexts each by tranversing the active bits in each structure.
2. Guess the 2^{m_b} possible m_b-bit subkey involved in E_b:
   (a) Initialized a list of 2^{m_f} counters corresponding to a m_f-bit subkey guess.
   (b) For each structure, partially encrypt plaintext P_1 under the guessed subkey bits in E_b to the position of α and decrypt the intermediate value xored the known difference α to obtain the plaintext P_2 in the same structure with P_1. Construct a set S with the relevant plaintexts and ciphertexts as

   $$S = \{(P_1, C_1, P_2, C_2) : E_b(P_1) \oplus E_b(P_2) = \alpha\}.$$

   (c) The size of S is $y \cdot 2^{r_b-1}$. Insert S into a hash table H indexed by the $n - r_f$ bits of C_1 and $n - r_f$ bits of C_2 that are 0 in δ'. We randomly choose (C_1, C_2) and (C_1', C_2') to generate quartet (C_1, C_2, C_3, C_4) with each $2(n - r_f)$-bit index, where $(C_3, C_4) = (C_1', C_2')$.

(d) Determine the key candidates related to E_f using the quartets obtained above and increase the corresponding counters. Similarly, denote the time complexity in this step as ϵ_3.
(e) Select the top 2^{m_f-h} hits in the counter to be the candidates, which delivers a h-bit or higher advantage.
(f) Guess the remaining $k-m_b-m_f$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key, where k is the key size.

The data complexity is $D = y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{\frac{n}{2}+1}/P$. The memory complexity is $D + D/2 + 2^{m_f}$ to store the data, set S and key counters. The time complexity to generate quartets by constructing set S is

$$T_1 = 2^{m_b} \cdot D = \sqrt{s} \cdot 2^{m_b+\frac{n}{2}+1}/P,$$

the complexity of determining the key candidates is

$$\begin{aligned} T_2 &= 2^{m_b} \cdot 2^{2(n-r_f)} \cdot 2 \cdot \binom{D^2 \cdot 2^{-2(n-r_f)-1}}{2} \cdot \epsilon_3 \\ &= 2^{m_b} \cdot D^2 \cdot 2^{2r_f-2n-2} \cdot \epsilon_3 \\ &= s \cdot 2^{m_b-n+2r_f}/P^2 \cdot \epsilon_3 \end{aligned}$$

and the exhaustive search complexity is

$$T_3 = 2^{m_b} \cdot 2^{m_f-h} \cdot 2^{k-(m_b+m_f)} = 2^{k-h},$$

where $h \le m_f$.

C.4 Algorithm 4: Dong *et al.*'s Single-key Variant

To avoid generating quartets that may never suggest key candidates as many as possible, Dong et al. presented a new rectangle attack framework [DQSW22] to transform Algorithm 3 into Algorithm 4 using fast filter with partially guessed key $k_f^{'}$ and h_f-bit inactive internal states resulted from partially guessed key. Denote $m'_f = |k_f^{'}|$. We summarize the procedures as follows:

1. Collect and store y structures of 2^{r_b} plaintexts each by tranversing the active bits in each structure.
2. Guess the possible $(m_b + m'_f)$-bit k_b and $k_f^{'}$ involved in E_b and part of E_f:
   (a) Initialize an array of $2^{m_f-m'_f}$ counters.
   (b) For each structure, construct set S in the same way with Model 3 in the following:

$$S = \{(P_1, C_1, P_2, C_2) : E_b(P_1) \oplus E_b(P_2) = \alpha\}.$$

   (c) The size of S is $y \cdot 2^{r_b-1}$. For each (P_1, C_1, P_2, C_2) in S, partially decrypt (C_1, C_2) to get two r'_f-bit partial internal state (Y_1, Y_2). Insert S into a hash table indexed by $n-r_f$ inactive bits of C_1, $n-r_f$ inactive bits of C_2, r'_f inactive bits of both Y_1 and Y_2.

(d) With each $2(n - r_f + r'_f)$-bit index, we pick two distinct (P_1, C_1, P_2, C_2), $(P_1^{'}, C_1^{'}, P_2^{'}, C_2^{'})$ to generate the quartet, denoted as (C_1, C_2, C_3, C_4), where $(C_3, C_4) = (C_1^{'}, C_2^{'})$.

(e) Determine the key candidates involved in E_f and increase the corresponding counters. Likewise, denote the time complexity in this step as ϵ_4.

(f) Select the top $2^{m_f - m'_f - h}$ hits in the counter to be the candidates, which delivers a h-bit or higher advantage.

(g) Guess the remaining $k - m_b - m_f$ unknown key bits according to the key schedule algorithm and exhaustively search over them to recover the correct key, where k is the key size.

The data complexity is $D = y \cdot 2^{r_b} = \sqrt{s} \cdot 2^{\frac{n}{2}+1}/P$. The memory complexity is $D + D/2 + 2^{m_f - m'_f}$ to store the data, set S and key counters. The time complexity of generating quartets by constructing set S is

$$T_1 = 2^{m_b + m'_f} \cdot D = \sqrt{s} \cdot 2^{m_b + m'_f + \frac{n}{2} + 1}/P,$$

the complexity to determine key candidates is

$$\begin{aligned} T_2 &= 2^{m_b + m'_f} \cdot 2^{2(n - r_f + r'_f)} \cdot 2 \cdot \binom{D \cdot 2^{-1-2(n-r_f+r'_f)}}{2} \cdot \epsilon_4 \\ &= 2^{m_b + m'_f} \cdot D^2 \cdot 2^{2r_f - 2r'_f - 2n - 2} \cdot \epsilon_4 \\ &= s \cdot 2^{m_b + m'_f - n + 2r_f - 2r'_f}/P^2 \cdot \epsilon_4 \end{aligned}$$

and the complexity of exhaustive search is

$$T_3 = 2^{m_b + m'_f} \cdot 2^{m_f - m'_f - h} \cdot 2^{k - (m_b + m_f)} = 2^{k-h},$$

where $h \leq 2^{m_f - m'_f}$.

Actually, there are also another two refinements of Algorithm 4 presented in [DQSW22]. The first refinement is to balanced the overall complexity by guessing different key cells among the partial key guess. The second is to apply the improved algorithm to related-key setting. More details about this can be find in [DQSW22]. In this paper, the refined algorithms can be grouped together, in which Algorithm 4 is a representative.