Hindawi
Security and Communication Networks
Volume 2022, Article ID 5360032, 11 pages
https://doi.org/10.1155/2022/5360032

Research Article

Multiple Impossible Differential Attacks for ForkAES

Zilong Jiang and Chenhui Jin

PLA SSF Information Engineering University, Zhengzhou 450000, China

Correspondence should be addressed to Zilong Jiang; dracipher@126.com

Received 30 September 2021; Revised 23 February 2022; Accepted 25 May 2022; Published 29 June 2022

Academic Editor: Jin Wook Byun

To yield a highly efficient authentication encryption design for very short messages, the tweakable forkcipher is proposed, which is a tweakable block cipher that uses forking construction to produce two output blocks. The designers also presented ForkAES, a forkcipher that is based on the round function of AES and the tweakable variant of KIASU. Therefore, the security of ForkAES is found on the block ciphers AES and KIASU. However, from the perspective of new forking construction, attackers may obtain some unique properties. Impossible differential attacks are widely used on block ciphers. This paper studies the security of ForkAES against multiple impossible differential cryptanalysis. Based on the property of the forking construction, two types of impossible differential distinguishers have been constructed. We first use the tweak with different truncated differences to build more attack trails. Then, we first propose the multiple impossible differential attack for ForkAES- $*$ -5-4. Thus, only a single round would remain as a security margin. Utilizing multiple attack trails, our attack scenario obtains more subkey bytes and enhances the subkey's sieving efficiency without increasing complexity. Furthermore, we carefully consider the process of recovering the master key, which can efficiently reject wrong candidate keys. In reconstruction queries, our attack reaches the longest number of rounds for ForkAES in impossible differential analysis, with $2^{100.8}$ lookups, $2^{85.8}$ chosen ciphertexts, and $2^{92.7}$ memory blocks to store AES states. In encryption queries, we improve the previous attacks on ForkAES by attacking one more round (i.e., ForkAES- $*$ -5-4), with $2^{118.2}$ lookups, $2^{111.4}$ plaintexts, and $2^{92.7}$ memory blocks to store AES states.

1. Introduction

Many authenticated encryption (AE) schemes need to perform the initialization or finalization phase to optimize the cost of handling long messages. However, in these general schemes, initialization or finalization raises the cost of handling short messages. For example, the TLS needs to produce a message authentication code (MAC) in the initialization phase. An efficient AE scheme for very short messages is an actively discussed topic. To yield a highly efficient AE design for very short messages, Andreeva et al. [1] proposed a tweakable forkcipher [1], which utilizes the forking construction to produce two output blocks from the same input. Compared with the TLS, the expanded ciphertexts of the forkcipher can also serve as the MAC, so there is no need to perform initialization to produce the MAC. Furthermore, the forking construction shares some computations, so the forkcipher is a highly efficient AE design for very short messages. Andreeva et al. [1] also proposed a forkcipher named ForkAES. Balli and Banik [2] analysed the hardware performance of ForkAES. They achieve a hardware implementation of ForkAES without any additional storage cost if the AES circuit can also perform decryption, thereby further optimizing the storage cost.

ForkAES is based on the round function AES [3] and the tweakable variant KIASU [4]. Andreeva et al. [1] briefly considered differential attacks, related-tweakey attacks, and meet-in-the-middle attacks. For other attacks, these authors stated that "the security of our forkcipher design can be reduced to the security of the AES and KIASU ciphers for further types of attacks." However, from the perspective of the new forking construction, attackers may obtain some unique properties. Reconstruction queries mean that attackers can choose one block ciphertext and obtain the other block ciphertext by reconstruction querying. Encryption queries mean that attackers obtain the two block ciphertexts

by encryption querying the plaintext. Banik et al. [5] introduced reflection differential trails and proposed some attacks of ForkAES- $*$ -4-4 ($*$ is any nonnegative integer). Bariant et al. [6] first presented truncated differential attacks on ForkAES- $*$ -5-5 (full ten rounds) in reconstruction queries, but the security of ForkAES in encryption queries needs to be further studied.

Impossible differential attack [7, 8] is a very powerful and important method of attacking block ciphers. For a long time, impossible differential attacks were among the most successful attacks for AES [9], KIASU [10] and Deoxys [11]. Tsunoo et al. [12] proposed the idea of multiple impossible differentials. Li et al. [13, 14] proposed the subkey early abort, and Boura et al. [15, 16] presented the state-test technique; using these techniques, these authors presented their successful attacks against classical block ciphers, such as Fox and AES.

This paper studies the security of ForkAES against multiple impossible differential cryptanalysis. Based on the property of the forking construction, two types of impossible differential distinguishers have been constructed. We first use the tweak with different truncated differences to construct more attack trails. Then, we propose the multiple impossible differential attack for ForkAES- $*$ -5-4. Utilizing multiple attack trails, our attack scenario obtains more subkey bytes and enhances the sieving efficiency of subkeys without increasing complexity. Furthermore, one of the impossible differential attacks needs to guess only five subkey bytes in paper [5], and the authors did not analyse the process of recovering the master key. Because the subkey bytes may not suffice, the cost of recovering the master key may be more than that of attacking subkey bytes. Compared with paper [5], we carefully consider the process of recovering the master key, which can reject wrong candidate keys efficiently. Compared with paper [6], we consider the attack scenario in encryption queries, which increases the understanding of ForkAES. To the best of our knowledge, we improve the previous attacks on ForkAES by attacking one more round in encryption queries. Table 1 summarizes some attacks of ForkAES.

This paper is organized as follows: in Section 2, we describe the ForkAES structure and some basic concepts. In Section 3, we construct six impossible differential distinguishers of ForkAES. In Section 4, we propose multiple impossible differential attacks on ForkAES- $*$ -5-4 and analyse our attack. Finally, Section 5 concludes this paper.

2. Preliminaries

2.1. Notations. We will introduce some basic notations used throughout this paper:

$P/C/T$: Plaintext/Ciphertext/Tweak;

$x_{i,(p,\ldots,r)}^{SB/SR/MCAK/AT}$: the intermediate state of the $p^{th}, \ldots, r^{th}$ bytes after SB/SR/MC/AK/AT in the i-th round;

Δx: the difference between x and x';

$k_{i,[p,\ldots,r]}$: the $p^{th}, \ldots, r^{th}$ bytes of subkey k_i;

$T(i)$: the i^{th} byte of tweak T;

$a \longrightarrow^{i-\text{round}} b$: the difference a cannot obtain the difference b through i round encryptions.

2.2. Brief Description of Impossible Differential Attack. To mount an impossible differential attack, the block cipher E is divided into three parts: $E = E_3 \circ E_2 \circ E_1$ (as shown in Figure 1). First, an impossible differential distinguisher $\Delta\text{ID}_{\text{in}} \nrightarrow \Delta\text{ID}_{\text{out}}$ through E_2 is constructed. Then, for the extra added rounds E_1 and E_3, the differential $\Delta\text{ID}_{\text{in}} \longrightarrow \Delta P_{\text{in}}$ through E_1^{-1} and $\Delta\text{ID}_{\text{out}} \longrightarrow \Delta C_{\text{out}}$ through E_3 occur with probability 1. The combination of the differential $\Delta\text{ID}_{\text{in}} \longrightarrow \Delta P_{\text{in}}$ and $\Delta\text{ID}_{\text{out}} \longrightarrow \Delta C_{\text{out}}$ is called an attack trail.

2.3. Brief Description of ForkAES. ForkAES-r_t-r_a-r_b means that the **internal state** is obtained after r_t rounds from plaintext; then, r_a and r_b denote the number of rounds from internal state to ciphertext C_0 and from internal state to ciphertext C_1, respectively. As shown in Figure 2, the original ForkAES is ForkAES-5-5-5. ForkAES encrypts the plaintext over the first five rounds to produce the internal state, and then the internal state is forked to produce two ciphertexts C_0 and C_1. We denote the internal state as S_5. The input state of the first branch is $x_5 = S_5$, and the output of the first branch is $x_{10} = C_0$. Similarly, the input state of the second branch is $y_5 = S_5$, and the output of the first branch is $y_{10} = C_1$.

The block size of ForkAES is 128 bits. As shown in Figure 3, the round function of ForkAES is similar to that of KIASU, which applies five operations as follows:

(1) SubBytes (SB)

This operation applies an 8-bit invertible S-box of AES on 16 bytes of the state.

(2) ShiftRows (SR)

The cyclic shift of each row is the same as AES.

(3) MixColumns (MC)

The multiplication of each column by the 4- × -4-cell MDS matrix of AES.

(4) AddRoundTweakey (AK)

This operation XORs with the round subkeys derived from the master key.

(5) AddRoundTweakey (AT)

This operation XORs with a tweak.

In this paper, k_0 represents the whitening key. SB^{-1}, SR^{-1}, MC^{-1}, AK^{-1}, and AT^{-1} represent the inverse operations of SB, SR, MC, AK, and AT, respectively. For more details on ForkAES, we can refer to the paper [1].

3. The Impossible Differential Distinguishers of ForkAES

The longest impossible differential key-independent distinguisher of AES is four rounds [17]. Utilizing the property

Table 1: The comparison of attacks on ForkAES.

Attack type	Rounds	Time		Data	Memory	Reference
		Encs.	MAs			
Reconstruction queries						
Rectangle diff.	ForkAES- $*$ -4-4	2^{28}	2^{35}	2^{35}	2^{33}	[5]
Impossible diff.	ForkAES- $*$ -4-4	2^{47}	2^{47}	$2^{39.4}$	2^{35}	[5]
Yoyo	ForkAES- $*$ -3-3	$2^{14.5}$	2^{29}	$2^{14.5}$	2^{29}	[5]
Imp.-diff. Yoyo	ForkAES- $*$ -4-4	$2^{128.83}$	–	$2^{122.83}$	$O(1)$	[5]
Impossible diff^{mk}	ForkAES- $*$ -5-4	$2^{7.2}$	$2^{100.8}$	$2^{85.8}$	$2^{92.7}$	This paper
Truncated diff.	ForkAES- $*$ -5-5	2^{125}	–	2^{119}	2^{83}	[6]
Encryption queries						
Rectangle	ForkAES- $*$ -4-4	$2^{88.5}$	$2^{92.4}$	2^{85}	$2^{86.4}$	[5]
Impossible diff.	ForkAES- $*$ -4-4	$2^{75.4}$	$2^{110.2}$	$2^{70.2}$	2^{110}	[5]
Impossible diff^{mk}	ForkAES- $*$ -5-4	$2^{7.2}$	$2^{118.2}$	$2^{111.4}$	$2^{92.7}$	This paper

mk: recover the master key; MAs: memory accesses.

of forking construction, we construct six 5-round impossible differential distinguishers of ForkAES.

The input difference of distinguisher Δx_8^{SR}: in the round function after the subkey k_8, the difference is after the SR operation.

The output difference of distinguisher Δy_5^{MC}: in the round function after the subkey k_{11}, the difference is after the MC operation.

The input difference Δx_8^{SR} (in the first branch) is mapped to the internal state difference and then obtains the output difference Δy_5^{MC} (in the second branch). The specific impossible differentials of six 5-round distinguishers are shown in Table 2. We construct two types of distinguishers by utilizing the tweak with a distinct truncated difference. The first type of distinguisher uses tweak pairs with only a 1-byte nonzero difference at position (0), and the second type of distinguisher uses tweak pairs with only a 1-byte nonzero difference at position (4). Figure 4 shows the sample of the first distinguisher. As shown in Figure 4, the impossible differential has been constructed in the first branch, and the output difference propagates to the internal state difference

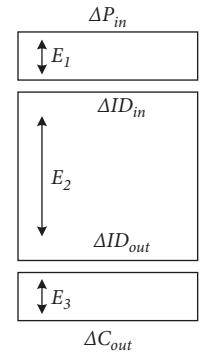

Figure 1: General structure of an impossible differential attack.

with probability 1, so the 5-round impossible differential distinguisher of ForkAES has been constructed.

Theorem 1. *As shown in Figure 4, there is a 5-round truncated impossible differential distinguisher of ForkAES:*

$$(*, *, *, 0, *, *, 0, *, *, 0, *, *, 0, *, *, *) \xrightarrow{5\text{–round}} (*, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0), \quad (1)$$

where $(*)$ *is the nonzero difference. The input difference is* Δx_8^{SR}*, and the output difference is* Δy_5^{MC}*. This distinguisher uses the tweak pairs with only a 1-byte nonzero difference at position (0).*

Proof. In the second branch, the output difference is Δy_5^{MC}. Through the MC^{-1}, SR^{-1}, and SB^{-1} operations, the internal state difference can be obtained with probability 1.

In the first branch, the difference of 4 bytes $\Delta x_{5,(0,1,2,3)}^{MC}$ can be any value, and the difference of the other 12 bytes of Δx_5^{MC} must be zero. Then, through the *SB*, *SR*, and *MC* operations, each column of difference Δx_6^{MC} is either fully inactive or fully active (Property 1).

From the difference Δx_8^{SR}, through the SR^{-1} and SB^{-1} operations, the rightmost column of Δx_7^{AT} is fully inactive, and the other three columns are fully active. Then, through the AT^{-1} operation, the difference of the byte at position (0) can be any value. Through the MC^{-1} operation, the rightmost column of Δx_7^{SR} is fully inactive, and the other three columns have at least one active byte. Through the SR^{-1}, SB^{-1}, and AT^{-1} operations, the difference Δx_6^{MC} exists in one column with at least one inactive byte and at least one active byte, thus contradicting Property 1. □

The other five impossible differential distinguishers of ForkAES could be proven similarly.

4. Multiple Impossible Differential Attacks on ForkAES- $*$ -5-4

Based on the two types of distinguishers in Section 3, we construct two attack trails with the same truncated

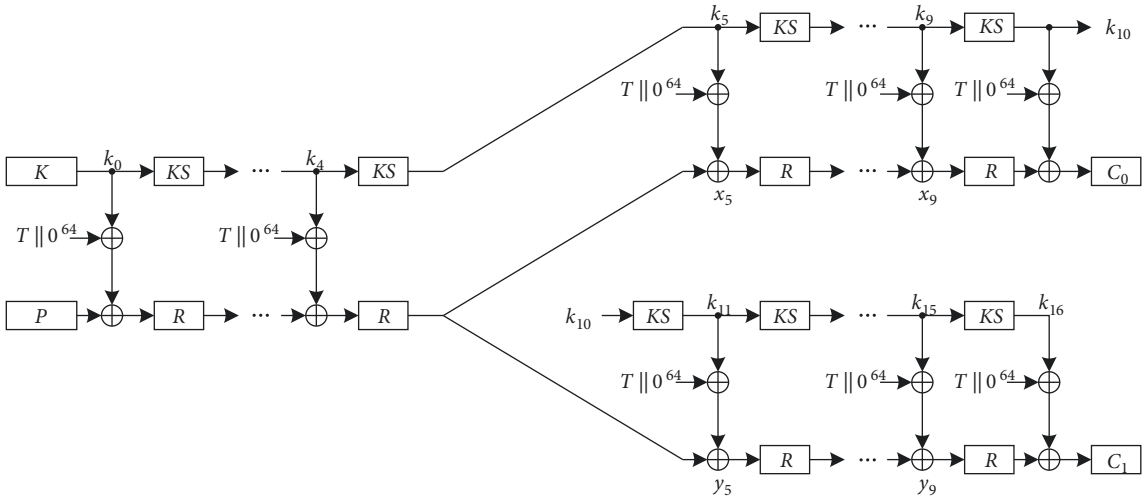

Figure 2: Illustration of ForkAES.

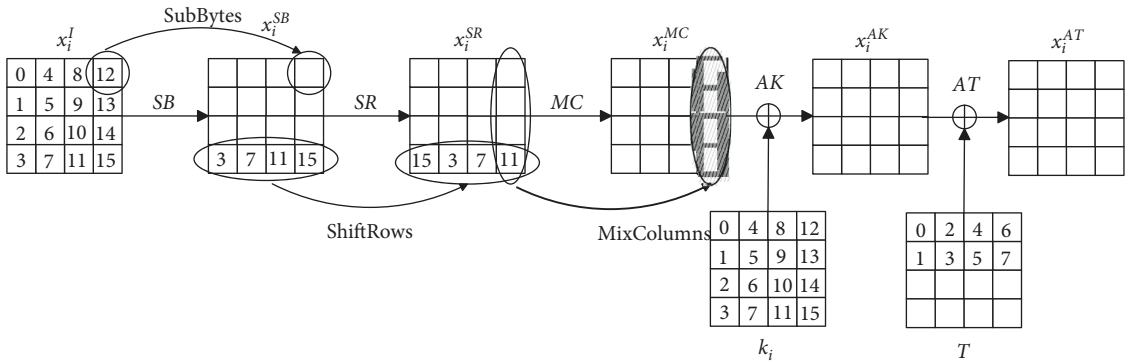

Figure 3: The round function of ForkAES.

difference of ciphertext C_0 pairs. We first propose the multiple impossible differential attack on ForkAES- $*$ -5-4. In consideration of the clear expression, we have $r_t = 5$. Our attack scenario can be applied to chosen ciphertext (reconstruction queries) and known plaintext (encryption queries).

As shown in Figure 5, we append two rounds in the first branch and four rounds in the second branch. We swap the order of the *SR*, *MC*, and *AK* operations of the third and fourth rounds in the second branch, and $\widehat{k}$ denotes the equivalent subkey. For the first attack trail, thirteen subkey bytes $(k_{10,(0,1,2,3,4,5,6,7)}, \widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$ need to be guessed. For the second attack trail, thirteen subkey bytes $(k_{10,(0,1,2,3,4,5,6,7)}, \widehat{k}_{13,(4)}, \widehat{k}_{14,(4,5,6,7)})$ need to be guessed. These two attack trails have common subkey bytes $k_{10,(0,1,2,3,4,5,6,7)}$, which can be repeatedly sieved to improve the result.

4.1. The Attack with Reconstruction Queries. The attack procedure includes the data-collection phase, precomputation phase, and key-recovery phase. In this section, we use the sieve method based on rapid sorting [18] to choose ciphertext pairs.

4.1.1. Data-Collection Phase. We choose the ciphertext C_0 and then obtain the ciphertext C_1 by reconstruction queries.

For the first attack trail, select 2^8 tweaks that take all possible values in $T_{(0)}$ and fix the other 7 bytes to constant values. Select 2^{64} ciphertexts C_0 that take all possible values in the 8 bytes at $(0, 1, 2, 3, 4, 5, 6, 7)$ and fix the other 8 bytes to constant values. We take $2^{72}(C_0, T_{(0)})$ as a structure. By reconstruction querying, through the AT^{-1}, MC^{-1}, and SR^{-1} operations, 2^{72} texts y_8^{AK} can be obtained. Using the sieve method based on rapid sorting, select $2^{72} \times (2^8 - 1)^9 \div 2 \times 2^{-96} \approx 2^{47}$ pairs of (y_8^{SB}, y'), where $\Delta y_{8,(0,1,2,3)}^{SB} \neq 0$ and the other 12-byte difference value is zero, and then store the ordered texts $(C_0, T_{(0)}, y_{8,(0,1,2,3)}^{SB})$ in Table Ω_1.

For the second attack trail, select 2^8 tweaks that take all possible values in $T_{(2)}$ and fix the other 7 bytes to constant values. Select the same 2^{64} ciphertexts C_0 in the first trail. These ciphertexts take all possible values in the 8 bytes at $(0, 1, 2, 3, 4, 5, 6, 7))$, and fix the other 8 bytes to constant values. We take $2^{72}(C_0, T_{(2)})$ as a structure. By reconstruction querying, through the AT^{-1}, MC^{-1}, and SR^{-1} operations, 2^{72} texts y_8^{AK} can be obtained. Using the sieve method based on rapid sorting, select $2^{72} \times (2^8 - 1)^9 \div 2 \times$

Table 2: Six 5-round truncated impossible differential distinguishers of ForkAES.

Type	Number	Zero input difference	Nonzero middle state difference	Nonzero tweak and output difference
First	1	(3, 6, 9, 12)	(0, 5, 10, 15)	(0)
	2	(2, 5, 8, 15)		
	3	(1, 4, 11, 14)		
Second	4	(3, 6, 9, 12)	(3, 4, 9, 14)	(4)
	5	(2, 5, 8, 15)		
	6	(0, 5, 10, 15)		

Note: the input difference Δx_8^{SR} propagates to the internal state with probability 0 and the output difference Δy_5^{MC} propagates to the internal state with probability 1.

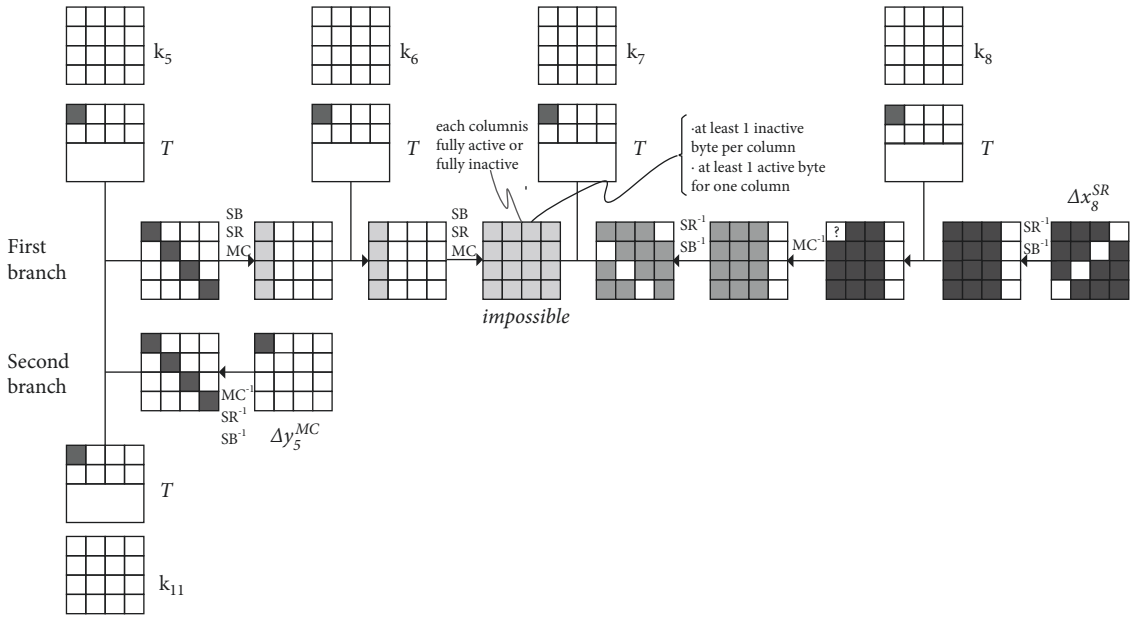

Figure 4: A sample of 5-round impossible differential distinguishers.

$2^{-96} \approx 2^{47}$ pairs of (y_8^{SB}, y'), where $\Delta y_{8,(4,5,6,7)}^{SB} \neq 0$ and the other 12-byte difference value is zero, and then store the ordered texts $(C_0, T_{(2)}, y_{8,(4,5,6,7)}^{SB})$ in Table Ω_2.

4.1.2. Precomputation Phase

Table Λ: let S, Δ_{in}, and Δ_{out} denote the S-box of ForkAES, the input difference and the output difference of the S-box, respectively. The equation $S(x) \oplus S(x \oplus \Delta_{\mathrm{in}}) = \Delta_{\mathrm{out}}$ has one solution on average when the values of the differences Δ_{in} and Δ_{out} are nonzero. We construct Table Λ to store the calculated value of x, indexed by all 2^{16} values of $\Delta_{\mathrm{in}}, \Delta_{\mathrm{out}}$.

Table H_1: for the first branch of the first attack trail, there are 3×2^{96} input difference values of three distinguishers. Through the MC operation, $3 \times 2^{96} \times 2^{-64} = 3 \times 2^{32}$ input difference values satisfy the demand for the truncated difference Δx_8^{AT}. We construct Table H_1 to store 3×2^{32} difference values of Δx_8^{MC}.

Table H_2: for the first branch of the second attack trail, there are 3×2^{96} input difference values of three distinguishers. Through the MC operation, $3 \times 2^{96} \times 2^{-64} = 3 \times 2^{32}$ input difference values satisfy the demand for the truncated difference Δx_8^{AT}. We construct Table H_2 to store 3×2^{32} difference values of Δx_8^{MC}.

Table H_3: for the second branch of the first attack trail, there are 2^8 difference values by guessing all possible values of $\Delta y_{7,(0)}^{SR}$. Through the MC operation, there are 2^8 difference values of $\Delta y_{7,(0,1,2,3)}^{MC}$. We construct Table H_3 to store these 2^8 difference values of $\Delta y_{7,(0,1,2,3)}^{MC}$.

Table H_4: for the second branch of the second attack trail, there are 2^8 difference values by guessing all possible values of $\Delta y_{7,(4)}^{SR}$. Through the MC operation, there are 2^8 difference values of $\Delta y_{7,(4,5,6,7)}^{MC}$. We construct Table H_3 to store these 2^8 difference values of $\Delta y_{7,(4,5,6,7)}^{MC}$.

4.1.3. Online Phase. In Steps 1 and 2, combining a series of techniques, use the first attack trail to reject some wrong subkeys. In Steps 3 and 4, use the second attack trail to reject some wrong subkeys for the remaining subkeys, thereby enhancing the efficiency of sieving subkeys. In Step 5, by comprehensively utilizing the subkeys information, use the master key recovering technique to reject wrong candidate

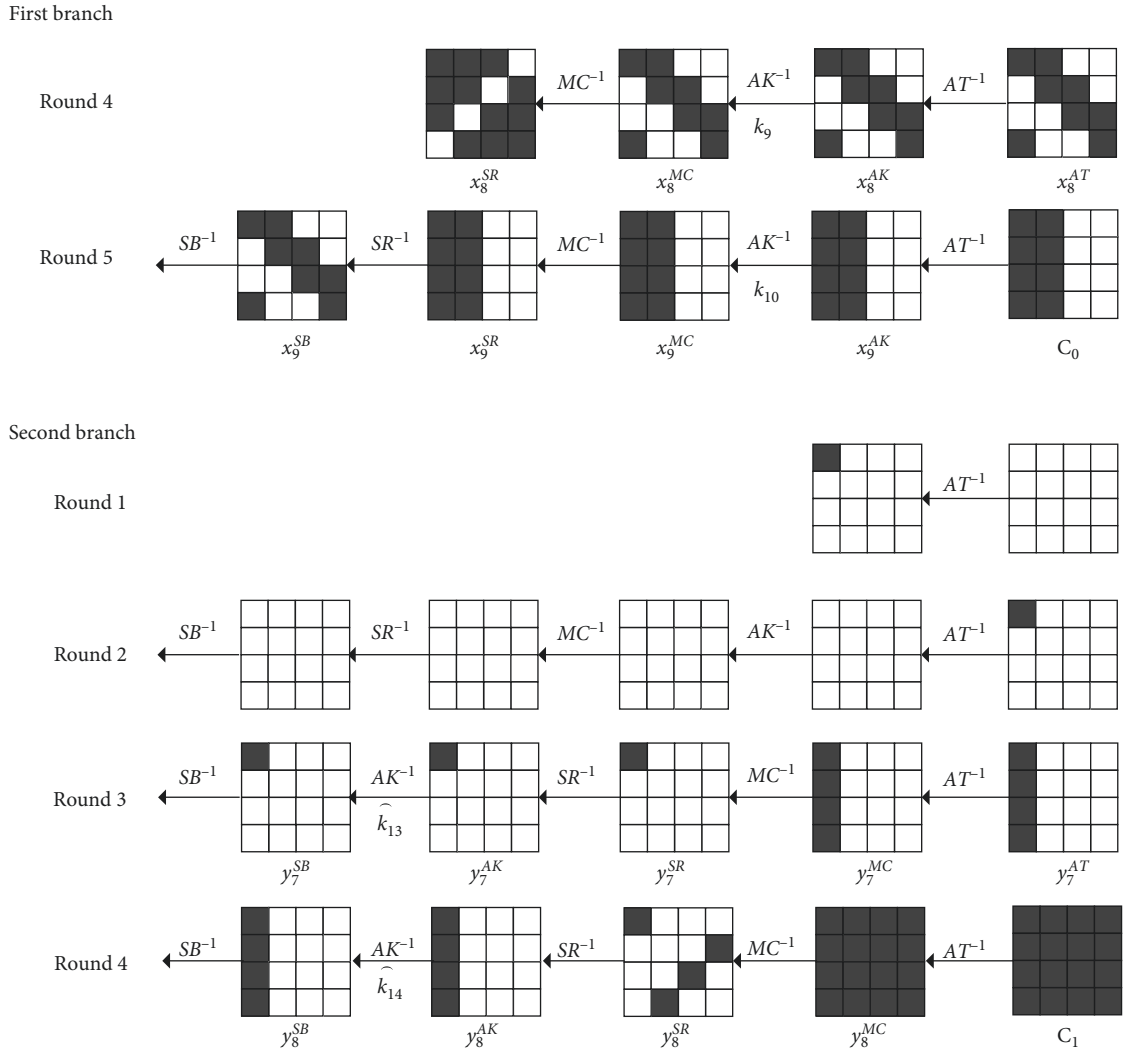

Figure 5: The first attack trail of ForkAES- $*$ -5-4.

keys until the right master key is obtained. We take 2^n structures, and the specific procedure is as follows:

(1) Use the first attack trail to calculate subkey $k_{10,(0,1,2,3,4,5,6,7)}$. Access Table Ω_1 to obtain 2^{n+47} pairs $(C_0 \oplus T_{(0)}, C_0' \oplus T'_{(0)})$. Through the AT^{-1}, AK^{-1}, MC^{-1}, and SR^{-1} operations, the 2^{n+47} output difference Δ_{out} can be obtained. Then, access Table H_1 to obtain the 3×2^{32} input difference Δ_{in}. Access Table Λ to obtain $3 \times 2^{n+79}$ subkeys $k_{10,(0,1,2,3,4,5,6,7)}$. Store the corresponding pairs $(y^{SB}_{8,(0,1,2,3)}, y')$ in Table U_1 indexed by subkey $k_{10,(0,1,2,3,4,5,6,7)}$, so $3 \times 2^{n+15}$ pairs remain on average.

(2) For the current subkey $k_{10,(0,1,2,3,4,5,6,7)}$, calculate subkey $(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$. Access Table H_3 to obtain 2^8 input difference $\Delta y^{MC}_{7,(0,1,2,3)}$. Access Table U_1 to obtain output difference.

Construct Table V_1, which has 2^{40} addresses indexed by 5-byte subkeys $(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$. For each address, set the initial value to 0. Then, set the counter that corresponds to F_1 with an initial value of 0. For each pair in Table U_1, 2^8 wrong subkeys $(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$ can be discarded by accessing Table Λ. If the value at the corresponding address of discarded subkey $(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$ is 0 in Table V_1, update the value to 1 and increase the value of F_1 by 1. If $F_1 = 2^{40}$, evaluate the current subkey $k_{10,(0,1,2,3,4,5,6,7)}$ as wrong, and discard it. Check all pairs in Table V_1; then, if still $F_1 < 2^{40}$, the value at the addresses of subkeys $(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$ is 0, and the corresponding subkeys $(k_{10,(0,1,2,3,4,5,6,7)}, \widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$ are candidates; store these $2^{40} - F_1$ candidates in Table W_1.

(3) Use the second attack trail to calculate the remaining subkey $k_{10,(0,1,2,3,4,5,6,7)}$. Access Table Ω_2 to obtain 2^{n+47} pairs $(C_0 \oplus T_{(2)}, C_0' \oplus T'_{(2)})$. Through the AT^{-1}, AK^{-1}, MC^{-1}, and SR^{-1} operations, the 2^{n+47} output difference Δ_{out} can be obtained. Then, Table H_2 is

accessed to obtain the 3×2^{32} input difference Δ_{in}. Access Table Λ to obtain $3 \times 2^{n+79}$ subkeys $k_{10,(0,1,2,3,4,5,6,7)}$. Store the corresponding pairs $(y^{SB}_{8,(4,5,6,7)}, y')$ in Table U_2, and index the pairs by subkey $k_{10,(0,1,2,3,4,5,6,7)}$, so $3 \times 2^{n+15}$ pairs remain on average.

(4) For the current subkey $k_{10,(0,1,2,3,4,5,6,7)}$, calculate subkey $(\widehat{k}_{13,(4)}, \widehat{k}_{14,(4,5,6,7)})$. Access Table H_4 to obtain 2^8 input difference $\Delta y^{MC}_{7,(4,5,6,7)}$. Access Table U_2 to obtain the output difference.

Construct Table V_2, which has 2^{40} addresses indexed by 5-byte subkeys $(\widehat{k}_{13,(4)}, \widehat{k}_{14,(4,5,6,7)})$. For each address, set the initial value to 0. Then, set the counter that corresponds to F_2 with an initial value of 0. For each pair in Table U_2, 2^8 wrong subkeys $(\widehat{k}_{13,(4)}, \widehat{k}_{14,(4,5,6,7)})$ can be discarded by accessing Table Λ. If the value at the corresponding address of discarded subkey $(\widehat{k}_{13,(4)}, \widehat{k}_{14,(4,5,6,7)})$ is 0 in Table V_2, update the value to 1 and increase the value of F_2 by 1. If $F_2 = 2^{40}$, evaluate the current subkey $k_{10,(0,1,2,3,4,5,6,7)}$ as wrong and discard it. Check all pairs in Table V_2. If still $F_2 < 2^{40}$, the value at addresses of subkeys $(\widehat{k}_{13,(4)}, \widehat{k}_{14,(4,5,6,7)})$ is 0, and the corresponding subkeys $(k_{10,(0,1,2,3,4,5,6,7)}, \widehat{k}_{13,(4)}, \widehat{k}_{14,(4,5,6,7)})$ are candidates; store these $2^{40} - F_2$ candidates in Table W_2.

(5) For $(k_{10,(0,1,2,3,4,5,6,7)}, \widehat{k}_{13,(0,4)}, \widehat{k}_{14,(0,1,2,3,4,5,6,7)})$, use the master key recovering technique to efficiently obtain the right master key. For each candidate, guess all 2^{64} subkeys $\widehat{k}_{14,(8,9,10,11,12,13,14,15)}$. Through the *SR* and *MC* operations, the subkey k_{14} can be obtained. Calculate $\widehat{k}_{13,(0,4)}$ with the help of the key schedule, and if the calculated value of $\widehat{k}_{13,(0,4)}$ differs from the value of the current subkey, evaluate the current candidate as wrong and discard it. Then, calculate $k_{10,(0,1,2,3,4,5,6,7)}$ with the help of the key schedule, and if the calculated value of $k_{10,(0,1,2,3,4,5,6,7)}$ differs from the value of the current subkey, evaluate the current candidate as wrong and discard it. Attackers check the remaining candidates by brute force until the right master key is obtained.

Then, we analyse the complexity of the attack with reconstruction queries.

The data-collection phase requires $2 \times 2^n \times 2^{72} \log_2(2^{72}) \approx 2^{n+79.2}$ comparisons and $2 \times 2 \times 2^{n+47} \times 13 \approx 2^{n+52.7}$ bytes of memory. The complexity of precomputation phase can be ignored compared with the data collecting phase.

The complexities of the online phase are as follows:

(1) Step 1 requires $2^{n+47} \times 3 \times 2^{32} \approx 2^{n+80.6}$ memory accesses (MAs) and $3 \times 2^{n+15} \times 2^{64} \times 2 \times 5 \approx 2^{n+83.9}$ bytes of memory.

(2) In Step 2, a wrong subkey $(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$ can pass the test of one pair with probability $1 - 2^{-24}$, and all 2^{40} wrong subkeys $(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$ cannot pass the test of 2^{24} pairs with probability $[1 - (1 - 2^{-24})^{2^{24}}]^{2^{40}} \approx e^{-2^{40-1.4425}}$. Therefore, the probability that all 2^{40} subkeys $(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$ can pass the test of $2^{24}d$ pairs but cannot pass the test of $2^{24}(d+1)$ pairs is $p_d \approx e^{-2^{40-1.4425(d+1)}} - e^{-2^{40-1.4425d}}$. Then, the mathematical expectation of d is

$$E(d) = \sum_{d=1}^{\infty} d\left[e^{-2^{40-1.4425(d+1)}} - e^{-2^{40-1.4425d}}\right] \approx 27.81 \approx 2^{4.8}. \tag{2}$$

Thus, Step 2 requires $2^{64} \times 2^{24} \times 2^8 \times 2^{4.8} \approx 2^{100.8}$ MAs. Wrong subkeys $k_{10,(0,1,2,3,4,5,6,7)}$ can pass the test of $3 \times 2^{n+15}$ pairs with the probability

$$p_n = 1 - \left[1 - \left(1 - 2^{-24}\right)^{2^{n+16.6}}\right]^{2^{40}} \approx 1 - e^{-2^{40-1.4425 \times 2^{n-7.4}}}. \tag{3}$$

Thus, $2^{64} \times p_n$ subkeys $k_{10,(0,1,2,3,4,5,6,7)}$ remain, and $2^{64} \times p_n \times 13$ bytes of memory are required to store subkeys $(k_{10,(0,1,2,3,4,5,6,7)}, \widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$.

(3) Similar to Step 1, Step 3 requires $2^{n+47} \times 3 \times 2^{32} \approx 2^{n+80.6}$ MAs and $3 \times 2^{n+15} \times 2^{64} \times 2 \times 5 \approx 2^{n+83.9}$ bytes of memory.

(4) In Step 4, some wrong subkeys $k_{10,(0,1,2,3,4,5,6,7)}$ have been discarded. As in Step 2, $2^{64} \times p_n \times 2^{24} \times 2^8 \times 2^{4.8} \approx 2^{100.8} \times p_n$ MAs are needed. Then, $2^{64} \times p_n^2$ subkeys $(k_{10,(0,1,2,3,4,5,6,7)}, \widehat{k}_{13,(4)}, \widehat{k}_{14,(4,5,6,7)})$ remain, so $2^{64} \times p_n^2 \times 18$ bytes of memory are needed.

(5) In Step 5, for each candidate $(k_{10,(0,1,2,3,4,5,6,7)}, \widehat{k}_{13,(0,4)}, \widehat{k}_{14,(0,1,2,3,4,5,6,7)})$, guessing all 2^{64} subkeys $\widehat{k}_{14,(8,9,10,11,12,13,14,15)}$ to calculate subkey $\widehat{k}_{13,(0,4)}$ requires $2^{128} \times p_n^2$ MAs. After sieving subkeys, $2^{112} \times p_n^2$ candidates remain. Calculating $k_{10,(0,1,2,3,4,5,6,7)}$ requires $2^{112} \times p_n^2 \times 4$ MAs. Then, $2^{48} \times p_n^2$ candidates remain. Then, the procedure by brute force requires $2^{48} \times p_n^2$ encryptions.

For the attack procedure with reconstruction queries, take $n = 12.8$, and then $p_n = 2^{-20.9}$ can be obtained. The time complexity and the memory complexity are dominated by Step 2 and Steps 1 and 3, respectively. Overall, this attack scenario requires $2^{85.8}$ chosen ciphertexts, $2^{100.8}$ lookups, $2^{7.2}$ encryptions and $2^{96.7}/16 = 2^{92.7}$ AES states.

According to the above analysis, the advantages of our attack scenario are as follows:

(1) The second attack trail is only applied for sieving the remaining subkeys. Because the subkeys $k_{10,(0,1,2,3,4,5,6,7)}$ are fewer, the time complexity of the attack scenario is not increased. Additionally, the second attack has high sieving efficiency. When $n = 12.8$, the probability that a wrong subkey $k_{10,(0,1,2,3,4,5,6,7)}$ can pass the test is $P_n = 2^{-20.9}$; that is,

Inputs: Array $L[p, \ldots, r]$; Value function VALUE$(L[i])$;
Outputs: Sorted array $L[p, \ldots, r]$ indexed the value of t byte $C_{(a^1,\ldots,a^t)}$

(1) IDQUICKSORT $(L, p, r,$ VALUE$(L[i]))$
(2) **if** $p < r$ **then**
(3) $\quad q \leftarrow$ IDPARTTION $(L, p, r,$ VALUE$(L[i]))$
(4) $\quad$ IDQUICKSORT $(L, p, q-1,$ VALUE$(L[i]))$
(5) $\quad$ IDQUICKSORT $(L, q+1, r,$ VALUE$(L[i]))$
(6) **end if**
(7) IDPARTTION $(L, p, r,$ VALUE$(L[i]))$
(8) $x \leftarrow$VALUE$(L[r])$
(9) $i \leftarrow p-1$
(10) **for** $j \leftarrow p$ to $r-1$ **do**
(11) $\quad$ **if** $x \geq$ VALUE$(L(r))$ **then**
(12) $\quad\quad i \leftarrow i+1$
(13) $\quad\quad$ exchange $L[i] \leftrightarrow L[j]$
(14) $\quad$ **end if**
(15) **end for**
(16) **return** $i+1$

Algorithm 1: Quicksort of plaintext-ciphertext pairs in impossible differential attack (IDQUICKSORT).

the second attack trail can discard $2^{20.9}$ more wrong subkeys without increasing the time complexity.

(2) Our attack scenario calculates five more subkey bytes without increasing the time complexity. In Step 5, the efficiency of sieving candidates can be improved because attackers obtain more subkey bytes to discard wrong candidates efficiently. Thus, our attack scenario can obtain a better result.

4.2. The Attack with Encryption Queries. The procedure of an attack with encryption queries is similar to the procedure of an attack with reconstruction queries. We omit the precomputation and online phases for the sake of brevity. This section elaborates mainly on the data-collection phase.

The idea of the data-collection phase is as follows: attackers obtain 2^m known plaintexts and guess a 1-byte tweak, and 2^{m+8} corresponding ciphertexts can be obtained by encryption queries. Then, the attackers select the pairs that meet the requirements of the attack trails. Thus, the attack scenario with encryption queries can be obtained.

Data-collection phase: from 2^m known plaintexts, the ciphertext (C_0, C_1) can be obtained by encryption queries.

For the first attack trail, attackers obtain 2^m known plaintexts, select 2^8 tweaks that take all possible values in $T_{(0)}$, and fix the other 7 bytes to constant values. Then, 2^{m+8} corresponding ciphertexts (C_0, C_1) can be obtained by encryption queries. For C_1, through the AT^{-1}, MC^{-1}, and SR^{-1} operations, (C_0, y_8^{AK}) can be obtained. Using the sieve method based on rapid sorting, select $2^{m+8} \times (2^{m+8}-1) \div 2 \times 2^{-160} \approx 2^{2m-145}$ pairs of (C_0, y_8^{AK}), where $\Delta y^{SB}_{8,(0,1,2,3)} \neq 0$ and $\Delta C^{SB}_{0,(0,1,2,3,4,5,6,7)} \neq 0$, and the difference value of the other 20 bytes is zero; then, store the ordered texts $(C_{0,(8,9,10,11,12,13,14,15)}, y^{SB}_{8,(0,1,2,3)})$ in Table Ω_3.

For the second attack trail, attackers use the same 2^m known plaintexts, select 2^8 tweaks that take all possible values in $T_{(2)}$, and fix the other 7 bytes to constant values. Then, 2^{m+8} corresponding ciphertexts (C_0, C_1) can be obtained by encryption queries. For C_1, through the AT^{-1}, MC^{-1}, and SR^{-1} operations, (C_0, y_8^{AK}) can be obtained. Using the sieve method based on rapid sorting, select $2^{m+8} \times (2^{m+8}-1) \div 2 \times 2^{-160} \approx 2^{2m-145}$ pairs of (C_0, y_8^{AK}), where $\Delta y^{SB}_{8,(4,5,6,7)} \neq 0$ and $\Delta C^{SB}_{0,(0,1,2,3,4,5,6,7)} \neq 0$ and the other 20 bytes difference value is zero; then, store the ordered texts $(C_{0,(8,9,10,11,12,13,14,15)}, y^{SB}_{8,(4,5,6,7)})$ in Table Ω_4.

Then, we analyse the complexity of the above attack with encryption queries.

The data-collection phase requires $2 \times 2^{m+8} \log_2 2^{m+8} \approx (m+8) \times 2^{m+9}$ comparisons, $2 \times 2 \times 2^{2m-145} \times 13 \approx 2^{2m-149.3}$ bytes of memory and 2^{m+9} known plaintexts.

For the attack with encryption queries, take $n = 12.8$, and $2^{n+47} = 2^{59.8}$ pairs meet the requirements of attack trails. For the attack with encryption queries, 2^{2m-145} pairs meet the requirements of attack trails, so $m = 102.4$. Thus, the time complexity is dominated by the data-collection phase, and the memory complexity is dominated by Steps 1 and 3. Overall, this attack scenario requires $2^{111.4}$ plaintexts, $2^{118.8}$ lookups, $2^{7.2}$ encryptions and $2^{96.7}/16 = 2^{92.7}$ AES states.

5. Conclusion

In this paper, we first utilize the tweak with different truncated differences to construct two types of attack trails of ForkAES and then propose the multiple impossible differentials of ForkAES- $*$ -5-4. Multiple attack trails obtain more subkey bytes and improve the efficiency of sieving keys without increasing the time complexity. Furthermore, we carefully consider the procedure of recovering the master key, which can efficiently reject wrong candidates. Combined with several technologies, the complexity of our attack scenarios can be optimized. In reconstruction queries, our attack reaches the longest number of rounds for ForkAES in

Inputs: 2^n arrays $L^1, \ldots, L^{2^n}$; 2^8 tweaks $T^1_{(0)}, \ldots, T^{2^8}_{(0)}$; Value fuction VALUE$(\Omega[i])$;
Outputs: 2^n sorted arrays $\Omega^1, \ldots, \Omega^{2^n}$ indexed the value of 12 bytes $y^{AK}_{8,(4,5,6,7,8,9,10,11,12,13,14,15)}$

(1) for $1 \le m \le 2^n$ do
(2) for $1 \le i \le 2^8$ do
(3) for $1 \le j \le 2^{64}$ do
(4) $y_8^{AK,(m,i,j)} \leftarrow$ CQPD$(T^i_{(0)}, C_0^{m,j})$
(5) $\Omega_1^m[(i-1) \times 2^{64} + j] \leftarrow (T^i_{(0)}, C_0^{m,j}, y_8^{AK,(m,i,j)})$
(6) end for
(7) end for
(8) IDQUICKSORT $(\Omega_1^m, 1, 2^{72}, \text{VALUE}(\Omega_1^m[i]))$
(9) end for

Algorithm 2: Sieved expected pairs of the first attack trail.

Input: Two arrays Ω_1, Ω_2; Four arrays $H_1, \ldots, H_4$; Array $\Lambda[\Delta\text{in}, \Delta\text{out}]$; Three arrays $V_1, \ldots, V_3$
Outputs: Right master key

(1) **for** all 2^{n+47} elements of array Ω_1 **do**
(2) $\Delta x_9^{SB,i} \leftarrow SR^{-1}[MC^{-1}[\Delta C_0^i \oplus \Delta T_0^i]]$
(3) **for** all 3×2^{32} elements $\Delta x_8^{MC,j}$ of array H_1 **do**
(4) $\Delta x_8^{AT} \leftarrow \Delta x_8^{MC,j} \oplus \Delta T_0^i$
(5) $x_9^{SB} \leftarrow \Lambda[\Delta x_8^{AT}, \Delta x_9^{SB,i}]$
(6) $k_{10,(0,1,2,3,4,5,6,7)} \leftarrow MC[SR[x_9^{SB}]] \oplus C_0^i \oplus T_0^i$
(7) $U_{1,k_{10,(0,1,2,3,4,5,6,7)}} \leftarrow$ the pair of $(T_0^i, y_8^{AK,i})$
(8) **end for**
(9) **end for**
(10) **for** all 2^{64} arrays $U_{1,k_{10,(0,1,2,3,4,5,6,7)}}$ **do**
(11) initialize variable $F_1 = 0$
(12) **for** all elements $(T_0^i, T_0'i, y_8^{AK,i}, y_8'AK, i)$ of array $U_{1,k_{10,(0,1,2,3,4,5,6,7)}}$ **do**
(13) **for** all 2^8 elements $\Delta y_7^{MC,j}$ of array H_3 **do**
(14) $\Delta y_7^{AT} \leftarrow \Delta y_7^{MC,j} \oplus \Delta T_0^i$
(15) $(y_7^{AT}, y_8^{SB}) \leftarrow \Lambda[\Delta y_7^{AT}, \Delta y_8^{AK}]$
(16) $\widehat{k}_{14,(0,1,2,3)} \leftarrow y^{SB}_{8,(0,1,2,3)} \oplus y^{AK}_{8,(0,1,2,3)}$
(17) $y_7^{SB} \leftarrow \Lambda[\Delta T_0^i, MC^{-1}[\Delta y_7^{MC,j}]]$
(18) $\widehat{k}_{13,(0)} \leftarrow y^{SB}_{7,(0)} \oplus y^{AK}_{7,(0)}$
(19) **if** $V_1[(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})] = 0$ **then**
(20) $V_1[(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})] + +$
(21) $F_1 + +$
(22) **end if**
(23) **end for**
(24) **if** $F_1 = 2^{40}$ **then**
(25) $V_3[k_{10,(0,1,2,3,4,5,6,7)}] + +$
(26) **Break**
(27) **end if**
(28) **end for**
(29) **end for**
(30) Based on array Ω_2, repeat step 1–9 to obtain $U_{2,k_{10,(0,1,2,3,4,5,6,7)}}$.
(31) Based on array V_3, utilize the similar step to discard wrong subkeys $(\widehat{k}_{13,(4)}, \widehat{k}_{14,(4,5,6,7)})$. Then the candidates $(k_{10,(0,1,2,3,4,5,6,7)}, \widehat{k}_{13,(0,4)}, \widehat{k}_{14,(0,1,2,3,4,5,6,7)})$ can be obtained.
(32) Guessing all 2^{64} values of $\widehat{k}_{14,(8,9,10,11,12,13,14,15)}$, the right master key can be obtained by brute force.

Algorithm 3: Multiple impossible differentials attack with reconstruction queries.

impossible difference analysis, with $2^{100.8}$ lookups, $2^{85.8}$ chosen ciphertexts, and $2^{92.7}$ memory blocks to store AES states. In encryption queries, we improve the previous attacks on ForkAES by attacking one more round, with $2^{118.2}$ lookups, $2^{111.4}$ plaintexts, and $2^{92.7}$ memory blocks to store AES states.

Appendix

Algorithm 1 is a general method to sieve the expected plaintext-ciphertext pairs in an impossible differential attack. Attackers need array $L[p, \ldots, r]$, and each element of $L[p, \ldots, r]$ is plaintext-ciphertext $(M^p, C^p), \ldots, (M^r, C^r)$. All plaintexts $(M^p, \ldots, M^r)$ satisfy the expected differential of the attack trail. Based on the expected differential of the attack trail, the differential value of t byte $C_{(a^1,\ldots,a^t)}$ is 0. Then, the corresponding value function VALUE$(L[i])$ is designed to obtain the value of t byte $C_{(a^1,\ldots,a^t)}$.

Algorithm 2 is proposed to sieve the expected ciphertext pairs of the first attack trail. There are 2^n structures $L^1, \ldots, L^{2^n}$, and each structure has 2^{64} values of ciphertexts $C_0^{i,1}, \ldots, C_0^{i,2^{64}}$ with the same value in the eight bytes of ciphertext $C_{0,(8,9,10,11,12,13,14,15)}$. Based on the ciphertext-tweak (C_0, T_0), state y_8^{AK} can be obtained through the reconstruction queries and part decryptions CQPD(C_0, T_0). Then, 2^n values of the plaintext-tweak-state (C_0, T_0, y_8^{AK}) are stored in the corresponding array Ω. The function VALUE$(\Omega[i])$ is proposed to take the value of the 12 bytes $y^{AK}_{8,(4,5,6,7,8,9,10,11,12,13,14,15)}$ of the i-th element $\Omega[i]$.

Utilizing Algorithm 2, attackers can obtain 2^n sorted arrays $\Omega_1^1, \cdots \Omega_1^{2^n}$. In each array, the plaintext-tweak-state pairs with the same value of the 12 bytes $y^{AK}_{8,(4,5,6,7,8,9,10,11,12,13,14,15)}$ are selected. Then, the pairs satisfy the expected differential of the first attack trail. We use the array Ω_1 to represent 2^n sorted arrays $\Omega_1^1, \cdots \Omega_1^{2^n}$ in consideration of clear expression. Additionally, 2^{n+47} expected plaintext-tweak-state pairs can be obtained from array Ω_1 (i.e., 2^n sorted arrays $\Omega_1^1, \cdots \Omega_1^{2^n}$). The array Ω_2 can be constructed by a similar algorithm for the second attack trail.

Algorithm 3 is the pseudocode of a multiple impossible differential attack with reconstruction queries. The corresponding 2^{n+47} expected values of plaintext-tweak-state pairs can be obtained by the two arrays Ω_1 and Ω_2. The calculated value of the S-box is stored in array $\Lambda[\Delta\text{in}, \Delta\text{out}]$. Side-by-side execution can be used when multiple S-boxes need to be accessed. The corresponding 3×2^{32} values of the expected differential Δx_8^{MC} are stored in arrays H_1 and H_2. The corresponding 2^8 values of the expected differential ΔY_7^{MC} are stored in arrays H_3 and H_4. Array V_1 (resp., V_2) is indexed by the value of five bytes $(\widehat{k}_{13,(0)}, \widehat{k}_{14,(0,1,2,3)})$ (resp., $(\widehat{k}_{13,(4)}, \widehat{k}_{14,(4,5,6,7)})$), and array V_3 is indexed by the value of eight bytes $k_{10,(0,1,2,3,4,5,6,7)}$. All elements of arrays V_1, V_2, and V_3 are set to 0.

Data Availability

The data are available at https://tosc.iacr.org/index.php/ToSC/search/index?query = Cryptanalysis+of+Forkciphers&dateFromYear = &dateFromMonth = &dateFromDay = &dateToYear = &dateToMonth = &dateToDay = &authors =

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work in this paper was supported by the Natural Science Foundation of China (grant no.: 61772547, 61902428, and 61802438).

References

[1] E. Andreeva, V. Lallemand, A. Purnal, R. A. Reyhanitabar, and D. Vizár, "Forkcipher: a new primitive for authenticated encryption of very short messages," in *Proceedings of the 25th International Conference on the Theory and Application of Cryptology and Information Security*, pp. 153–182, Kobe, Japan, December 2019.

[2] F. Balli and S. Banik, "Exploring lightweight efficiency of ForkAES," in *Proceedings of the, 20th International Conference on Cryptology in India*, pp. 514–534, Hyderabad, India, December 2019.

[3] J. Daemen and V. Rijmen, *The Design of Rijndael - the Advanced Encryption Standard (AES)*Information Security and Cryptography, Beijing, China, 2020.

[4] J. Jean, I. Nikolić, and T. Peyrin, "Tweaks and keys for block ciphers: the TWEAKEY framework," in *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*, pp. 153–182, aiwan, China, December 2014.

[5] S. Banik, J. Bossert, A. Jana et al., "Cryptanalysis of forkAES," *Applied Cryptography and Network Security*, vol. 11464, pp. 43–63, 2019.

[6] A. Bariant, N. David, and G. Leurent, "Cryptanalysis of Forkciphers," in *Proceedings of the IACR Transactions on Symmetric Cryptology*, pp. 233–265, Beijing, China, March 2020.

[7] E. Biham, A. Biryukov, and A. Shamir, "Cryptanalysis of skipjack reduced to 31 rounds using impossible differentials advances in cryptology - EUROCRYPT," in *Proceedings of the 99, international conference on the theory and application of cryptographic techniques*, pp. 12–23, Prague, Czech Republic, May 1999.

[8] K. Aoki, T. Ichikawa, M. Kanda et al., "Camellia: a 128-bit block cipher suitable for multiple platforms - design and analysis," in *Proceedings of the Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000*, pp. 39–56, Waterloo, Canada, August 2000.

[9] H. Mala, M. Dakhilalian, V. Rijmen, and M. Modarres-Hashemi, "Improved impossible differential cryptanalysis of 7-round AES-128," in *Proceedings of the 11th International Conference on Cryptology in India*, pp. 282–291, Hyderabad, India, December 2010.

[10] C. Dobraunig and E. List, "Impossible-differential and boomerang cryptanalysis of round-reduced kiasu-bc," in *Proceedings of the Topics in Cryptology-CT-RSA 2017-The Cryptographers' Track at the RSA Conference 2017*, pp. 207–222, San Francisco, CA, USA, February 2017.

[11] R. Zong, X. Dong, and X. Wang, "Related-tweakey impossible differential attack on reduced-round Deoxys-BC-256," *Science*

China Information Sciences, vol. 62, no. 3, p. 12, Article ID 32102, 2019.

[12] Y. Tsunoo, E. Tsujihara, M. Shigeri, and T. T. H. Saito, "Impossible differential cryptanalysis of CLEFIA," in *Proceedings of the 15th International Workshop, FSE 2008, Lausann*, pp. 398–411, Fast Software Encryption, Lausanne, Switzerland, February 2008.

[13] X. Li, F.-W. Fu, and X. Guang, "Multiple impossible differential cryptanalysis on reduced FOX," *IEICE - Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E98.A, no. 3, pp. 906–911, 2015.

[14] X. Li, C. H. Jin, and F. W. Fu, "Improved results of impossible differential cryptanalysis on reduced FOX," *The Computer Journal*, vol. 59, no. 4, pp. 541–548, 2016.

[15] C. Boura, M. Naya-Plasencia, V. Suder, and T. R. O. C. Kaoshiung, "Scrutinizing and improving impossible differential attacks: applications to clefia, camellia, lblock and simon," in *Proceedings of the 20th International Conference on the Theory and Application of Cryptology and Information Security*, pp. 179–199, Taiwan, China, December 2014.

[16] C. Boura, V. Lallemand, M. Naya-Plasencia, and V. Suder, "Making the impossible possible," *Journal of Cryptology*, vol. 31, no. 1, pp. 101–133, 2018.

[17] T. Cui, C. Jin, B. Zhang, and Z. G. Chen, "Searching all truncated impossible differentials in SPN," *IET Information Security*, vol. 11, no. 2, pp. 89–96, 2017.

[18] Q. G. Zhang, "Plaintext pair sieve methods in impossible differential attack," *Computer Engineering*, vol. 36, pp. 127–129, 2010.