IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2024, No. 2, pp. 85–117. DOI:10.46586/tosc.v2024.i2.85-117

Key Recovery, Universal Forgery, and Committing Attacks against Revised Rocca: How Finalization Affects Security

Ryunouchi Takeuchi[1], Yosuke Todo[2] and Tetsu Iwata[1]

[1] Nagoya University, Nagoya, Japan
takeuchi.ryunosuke.u2@s.mail.nagoya-u.ac.jp,tetsu.iwata@nagoya-u.jp
[2] NTT Social Informatics Laboratories, Musashino, Japan
yosuke.todo@ntt.com

Abstract. This paper examines the security of Rocca, an authenticated encryption algorithm designed for Beyond 5G/6G contexts. Rocca has been revised multiple times in the initialization and finalization for security reasons. In this paper, we study how the choice of the finalization affects the overall security of Rocca, covering key recovery, universal forgery, and committing attacks. We show a key-recovery attack faster than the exhaustive key search if a linear key mixing is used in the finalization. We also consider the ideally secure keyed finalization, which prevents key-recovery attacks. We show that, in the nonce-misuse setting, this does not prevent universal forgery with a practical data complexity, although the time complexity is high. Our result on committing attacks shows that none of the versions of Rocca considered in this paper is secure. We complete our analysis by presenting a concrete example of colliding inputs against the designers' latest version of Rocca in the FROB setting, a strong notion of the committing security. Our analysis significantly improves the key committing attack against Rocca shown in ToSC 2024(1)/FSE 2024.

Keywords: Rocca · key recovery · universal forgery · committing attacks

1 Introduction

Background. In the era of digitalization, with the rapid exchange of vast data volumes, the advent of Beyond 5G/6G necessitates an encryption scheme that is both efficient and secure. To realize such schemes, leveraging the AES [AES01] round function is a promising approach since modern CPUs have AES-NI [Gue10] instruction set to accelerate the encryption with the block cipher AES. Ciphers like AEGIS [WP13] and Tiaoxin-346 [Nik14] are examples employing SIMD instructions and the AES round function. Further efficiency improvement is observed in the round function identified by Jean and Nikolić [JN16], outperforming AEGIS and Tiaoxin-346. Rocca [SLN$^+$21], as introduced by Sakamoto et al. in ToSC 2021(2)/FSE 2022, stands out as an authenticated encryption designed for Beyond 5G/6G contexts. It also leverages the AES round function and achieves promising speed and security.

Rocca is a nonce-based authenticated encryption with associated data (AEAD) scheme, i.e., it provides both privacy and authenticity of messages, and authenticity of associated data, with the use of a nonce. A nonce is a non-repeating value that is used to securely encrypt multiple messages, and the use of a nonce has critical importance in its security. Yet, its implementation, as evidenced in systems like WPA [VP17] and VoLTE [RKHP20], reveals challenges, with instances of nonce repetition documented, highlighting a significant security concern.

Received: 2024-03-01 Accepted: 2024-05-01 Published: 2024-06-18

Table 1: Comparison with several versions of Rocca

Version	Key absorption		Reference
	End of init.	Beginning of fin.	
Rocca-v1	✗	✗	[SLN+21]
Rocca-v2	✓	✓	[HII+22]
Rocca-v3	✓	✓	[SLN+22a]
Rocca-v4	✓	✗	[SLN+22b]

In ToSC 2022(3)/FSE 2023, Hosoyamada et al. showed vulnerabilities in Rocca [HII+22]. They first showed a state-recovery attack that recovers the internal state of Rocca and works practically in the nonce-misuse setting, in which the attacker can repeat using the same nonce multiple times. In the nonce-respecting setting, the state-recovery attack works by making 2^{128} decryption queries. Once the internal state is recovered, the structure of Rocca allows the attacker to recover the secret key. Rocca has a key of 256 bits, and this breaks the designers' security claim regarding key recovery. Hosoyamada et al. also suggested countermeasures, and one of them is to XOR the key at the end of the initialization and at the beginning of the finalization. To make it distinct from the original Rocca, we call the original version Rocca-v1, and the version by Hosoyamada et al. Rocca-v2. The countermeasure of Rocca-v2 cannot prevent the state-recovery attack itself. However, XORing the key at the end of the initialization counters the trivial key-recovery attack after the state-recovery attack. XORing the key at the beginning of the finalization counters the practical universal forgery attack after the state-recovery attack.

In response to Hosoyamada et al.'s attack, the designers of Rocca updated the ePrint version [SLN+22a] and adopted XORing the key at the end of the initialization and at the beginning of the finalization. However, the subsequent ePrint version [SLN+22b] saw the removal of XORing the key at the beginning of the finalization, raising security concerns. We call the version in [SLN+22a] Rocca-v3, and the one in [SLN+22b] Rocca-v4. See Table 1 for the versions of Rocca. This alteration made in Rocca-v4 maintains security against the trivial key-recovery attack. However, under the nonce-misuse setting, it allows a practical universal forgery attack.

It turns out that the choice of the finalization affects the overall security of Rocca. This paper delves into the security implications of various versions of Rocca, focusing on the risks associated with omitting XORing the key in the finalization phase, or more generally, we study how the choice of the finalization function affects the overall security of Rocca and clarify security trade-offs.

1.1 Our Contributions

We consider various versions of Rocca and present the security analysis in terms of key recovery, universal forgery, and committing security. Rocca-v1, -v2, -v3, and -v4 all claim 256-bit security against key-recovery attacks and 128-bit security against forgery attacks in the nonce-respecting setting, where the key length of these scheme are all 256 bits. No security claim is made in the nonce-misuse setting, in the related-key setting, and in the known-key setting. We consider not only four existing versions of Rocca but also ones using an arbitrary linear key expansion in the finalization or an ideally secure keyed finalization.

Key Recovery Attacks on Rocca-v2 and Rocca-v3. We start by analyzing the security of versions of Rocca in [SLN+22a] and in [HII+22] with respect to key-recovery attacks. The state-recovery attack from [HII+22] still works on Rocca-v2 and Rocca-v3, and XORing the key at the end of the initialization prevents a trivial key-recovery attack. The key XORed at the beginning of the finalization is supposed to prevent forgery attacks. However, we fully exploit the key XORed at the finalization to mount a key-recovery attack on both Rocca-v2 and Rocca-v3. Our attack needs to run the state-recovery attack from [HII+22] whose complexity is 2^{20} in the nonce-misuse setting, and 2^{128} in the nonce-respecting setting.

As a result, we show a key-recovery attack against Rocca-v2 and Rocca-v3 with a complexity of 2^{128} in the nonce-misuse setting, and 2^{192} in the nonce-respecting setting.

Key Recovery and Universal Forgery Attacks on Rocca with Stronger Finalization. In light of the key-recovery attacks against Rocca-v2 and Rocca-v3, a plausible countermeasure would be to strengthen the finalization by applying a linear mapping over the key before XORing it into the state. We call it "Rocca + any linear key mixing." We show that this still allows a key-recovery attack faster than the exhaustive key search. In order to substantiate the attack, we design an algorithm to derive a message that interpolates two internal states, i.e., given two internal states S and S', our state-interpolation algorithm returns a message M such that starting from S, after absorption of M, the internal state becomes S'. The state-interpolation algorithm works with an offline complexity of 2^{160}, and our key-recovery attack uses the state-interpolation algorithm as a subroutine. The overall complexity of our key-recovery attack is 2^{208} in the nonce-respecting setting.

It turns out that none of the linear key expansion prevents key-recovery attacks, and then using the ideally secure keyed finalization is an option to avoid key-recovery attacks. We call it "Rocca + ideal keyed finalization." The key recovery is no longer trivial, and we continue analyzing its security in terms of universal forgery, which is the strongest notion of authenticity. We analyse this in the nonce-misuse setting. This setting has its significance, as GCM [MV04] allows universal forgery attacks if a nonce is repeated [Jou06], which is a practical concern [BZD+16]. In fact, its vulnerability in the nonce-misuse setting is one of the reasons of the initiation of CAESAR [CAE], and is the starting point of the efforts to build various robust AEAD schemes in the nonce-misuse setting, see e.g., [GL15, PS16, ADL17].

We show that, even with the ideally secure keyed finalization, this still allows a universal forgery attack in the nonce-misuse setting with a practical data complexity by making use of the state-interpolation algorithm, while the time complexity is impractically high for the use of the state-interpolation algorithm.

Committing Attacks. We next consider the security of Rocca with respect to committing security, which is the notion formalized for AEAD by Farshim et al. [FOR17] under the name of robust AE. The notion requires the infeasibility of an attacker to find two distinct inputs that result in the same output. It has been observed that various real world applications require this property, e.g., in end-to-end encrypted message systems [GLR17, DGRW18], key rotation in key management services, envelope encryption, and Subscribe with Google (SwG) [ADG+22], and Shadowsocks proxy servers and password-authenticated key exchange (PAKE) [LGR21].

There are various security notions on the committing security [FOR17, CR22, BH22]. Let (K, N, A, M) be the input of AEAD, which is the key, nonce, associated data, and the message, respectively, and let $(C, T) = \mathrm{Enc}_K(N, A, M)$ be the output, where C is the ciphertext and T is the tag. We write $\mathrm{Dec}_K(N, A, C, T) = M$ or $\mathrm{Dec}_K(N, A, C, T) = \perp$ for its decryption. Following [BH22], the goal of an attacker is to find distinct (K, N, A, M) and (K', N', A', M') such that $\mathrm{Enc}_K(N, A, M) = \mathrm{Enc}_{K'}(N', A', M')$. In more detail, the distinctness condition requires $(K, N, A, M) \neq (K', N', A', M')$ in CMT-4 notion, $(K, N, A) \neq (K', N', A')$ in CMT-3 notion, and $K \neq K'$ in CMT-1 notion. We also

Table 2: Complexity of key recovery and committing attacks against different versions of Rocca. The key length is 256 bits for all the versions. $O(1)$ denotes computation of a few encryption.

	key recovery		committing attack		
Version	nonce-misuse	nonce-respect	FROB	CMT-2	CMT-3
Rocca-v1 [SLN+21]	2^{20} [HII+22]	2^{128} [HII+22]	2^{16}	2^{16}	$O(1)$
Rocca-v2 [HII+22]	2^{128}	2^{192}	-	2^{16}	$O(1)$
Rocca-v3 [SLN+22a]	2^{128}	2^{192}	-	2^{16}	$O(1)$
Rocca-v4 [SLN+22b]	-	-	2^{16}	2^{16}	$O(1)$
Rocca + any linear key mixing	2^{208}	2^{208}	-	2^{16}	$O(1)$
Rocca + ideal keyed finalization	-	-	-	2^{16}	$O(1)$

consider the strong notion called FROB notion (full robustness notion) [FOR17]. The notion is adapted to AEAD in [GLR17], and following [GLR17, MLGR23], the goal of the adversary is to output (K, N, A), (K', N', A'), and (C, T) such that $\mathrm{Dec}_K(N, A, C, T) \neq \perp$, $\mathrm{Dec}_{K'}(N', A', C, T) \neq \perp$, $K \neq K'$, and $N = N'$. CMT-4 and CMT-3 notions are equivalent, CMT-3 security implies CMT-1 security, and CMT-1 security implies FROB security [BH22].

The committing security is studied in [DFI+24], covering AEGIS [WP13], Rocca-S [ABC+23], Tiaoxin-346 [Nik14], and Rocca (Rocca-v1). The result on Rocca-v1 directly applies to Rocca-v4, and is the FROB attack with a complexity of 2^{128}. This exceeds the generic complexity of 2^{64}, which is the birthday bound of the tag length, concluding that Rocca-v1 has proven resistance to the attack presented in [DFI+24], and posing its committing security as an open question.

We start by analysing CMT-3 security, focusing on the case $K = K', N = N'$, and $A \neq A'$. By using the 7-round differential trail from zero to zero state difference shown in [HII+22], we show a CMT-3 attack with a practical complexity. The attack works regardless the choice of the finalization. In particular, we present a concrete example of colliding inputs of Rocca-v3 [SLN+22a], practically breaking its committing security.

We next consider a CMT-2 attack, which is not formally defined in [BH22], but is naturally defined as $(K, N) \neq (K', N')$ as the distinctness condition. We consider this notion as the attack can be converted into CMT-1 and FROB attacks for schemes with non-keyed finalization, i.e., Rocca-v1 and Rocca-v4. Starting from two distinct states, we develop an algorithm to output distinct associate data such that after the absorption, the resulting states collide. The complexity of the algorithm is 2^{16}. As a result, we have a CMT-2 attack against all the versions of Rocca, and we also have FROB and CMT-1 attacks against Rocca-v1 and Rocca-v4. We experimentally verify the correctness of the attack by presenting a concrete example of FROB attack against Rocca-v4. This result significantly improves the FROB attack of 2^{128} complexity against Rocca-v1 in [DFI+24] by a factor of 2^{112} into a practical one, and negatively solves the open question in [DFI+24].

See Tables 2 and 3 for the summary of our results.

1.2 Related Works

Rocca has already received various security analyses. Anand and Isobe evaluated the security against differential fault attacks in the nonce-misuse setting [AI21]. This attack results in a complete internal state recovery by injecting 4×48 faults. Bonnetain and Schrottenloher analyzed quantum state-recovery attack in Q2 setting [BS23]. See [AI23] for the implementation of the quantum circuit of Rocca. Shiraya et al. analyzed distinguishing attacks on the initialization phase and keystream [STSI23]. Takeuchi et al. studied the optimality of the round function of Rocca [TSI23a]. See [TSI23b] for the security evaluation

Table 3: Complexity of universal forgery attacks against different versions of Rocca. In the column of Version, "non-keyed finalization" refers to Rocca-v1 and Rocca-v4, and "keyed finalization" refers to Rocca-v2, Rocca-v3, Rocca + any linear key mixing, and Rocca + ideal keyed finalization. Results on "non-keyed finalization" directly follow from [HII+22]. $O(1)$ denotes a few encryption queries.

	nonce-misuse		nonce-respect		
Version	data	time	data	time	Ref.
non-keyed finalization	2	2^{20}	2^{128}	2^{128}	[HII+22]
keyed finalization	2^{128}	2^{128}	2^{128}	2^{128}	Generic complexity
	$O(1)$	2^{160}			Sect. 4.3.2

of the initialization of Rocca.

Organization of This Paper. In Sect. 2, we review the specification of Rocca. An overview of the state-recovery in [HII+22] is given in Sect. 2.3. In Sect. 3, we present key-recovery attacks against Rocca-v2 and Rocca-v3. In Sect. 4, we present the state-interpolation algorithm and use the algorithm for key recovery against Rocca + any linear key mixing and for nonce-misuse universal forgery against Rocca + ideal keyed finalization. We cover committing attacks in Sect. 5, and conclude the paper in Sect. 6. Appendices A and B show test cases of our committing attacks.

2 Preliminaries

In this section, we introduce the notation and present the specification of various versions of Rocca. The original version was proposed at ToSC 2021(2)/FSE 2022 [SLN+21], and we call it Rocca-v1 to make it distinct from other versions. At ToSC 2022(3)/FSE 2023 [HII+22], Hosoyamada et al. presented attacks on Rocca-v1 and broke the security claim. In the same paper, Hosoyamada et al. suggested tweaking the specification to XORing the secret key at the end of the initialization and at the beginning of the finalization. We call the version Rocca-v2. A similar tweak was adopted in the revised ePrint version of Rocca [SLN+22a]. We call the version Rocca-v3. Later, the ePrint version was again updated, where absorbing the secret key at the beginning of the finalization was removed [SLN+22b]. We call the version Rocca-v4. Table 1 summarizes the comparison of several versions of Rocca.

2.1 Specification of Rocca-v1 [SLN+21]

Notation. A block is defined as a 16-byte value. A block can be represented as a 4×4 byte state matrix, expressed as

$$X = \begin{bmatrix} X_{0,0} & X_{0,1} & X_{0,2} & X_{0,3} \\ X_{1,0} & X_{1,1} & X_{1,2} & X_{1,3} \\ X_{2,0} & X_{2,1} & X_{2,2} & X_{2,3} \\ X_{3,0} & X_{3,1} & X_{3,2} & X_{3,3} \end{bmatrix}.$$

Each $X_{i,j}$ represents one byte of data. The state of Rocca, denoted as S, is composed of 8 blocks as $S = (S[0], S[1], \ldots, S[7])$, where for $0 \leq i \leq 7$, $S[i]$ is a 128-bit string. The state at time t is written as $S_t = (S_t[0], S_t[1], \ldots, S_t[7])$. Two 128-bit constant blocks Z_0 and Z_1 are defined as

$$\begin{cases} Z_0 = \texttt{0x428A2F98D728AE227137449123EF65CD}\,, \\ Z_1 = \texttt{0xB5C0FBCFEC4D3B2FE9B5DBA58189DBBC}\,. \end{cases} \quad (1)$$

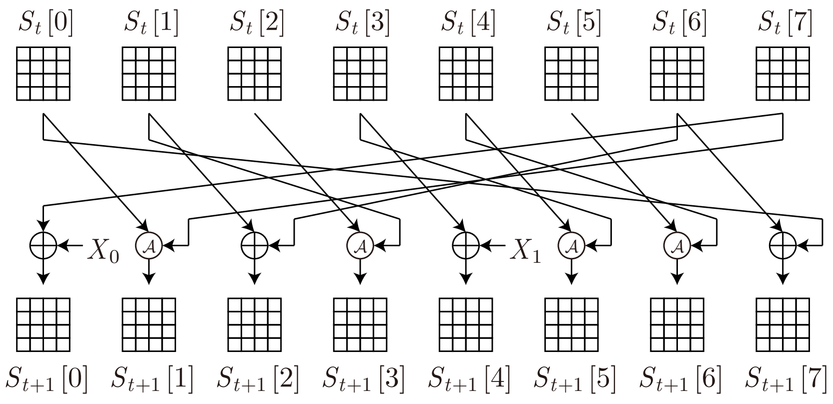

Figure 1: The round function of Rocca

The function $\mathcal{A}(X)$ represents the AES round function without AddRoundKey, and is defined as

$$\mathcal{A}(X) = \text{MixColumns} \circ \text{ShiftRows} \circ \text{SubBytes}(X)\,,$$

where MixColumns, ShiftRows, and SubBytes are the operations as defined in the specification of the AES [AES01]. These operations are written as MC, SR, and SB, respectively. For a byte $X \in \{0,1\}^8$, $\text{Sb}(X)$ denotes the output of the AES S-box on X.

Additionally, $|X|$ denotes the length of a bit string X in bits, 0^l is the zero string of length l bits, and $X \parallel Y$ represents the concatenation of bit strings X and Y. Finally, $R(S_t, X_0, X_1)$ denotes the round function used to update the state S_t, where X_0 and X_1 are two input blocks.

The Round Update Function. The input of the round function $R(S_t, X_0, X_1)$ of Rocca consists of the state S_t and two blocks X_0 and X_1. The output $S_{t+1} \leftarrow R(S_t, X_0, X_1)$ is computed as follows:

$$\begin{cases} S_{t+1}[0] = S_t[7] \oplus X_0 \\ S_{t+1}[1] = \mathcal{A}(S_t[0]) \oplus S_t[7] \\ S_{t+1}[2] = S_t[1] \oplus S_t[6] \\ S_{t+1}[3] = \mathcal{A}(S_t[2]) \oplus S_t[1] \\ S_{t+1}[4] = S_t[3] \oplus X_1 \\ S_{t+1}[5] = \mathcal{A}(S_t[4]) \oplus S_t[3] \\ S_{t+1}[6] = \mathcal{A}(S_t[5]) \oplus S_t[4] \\ S_{t+1}[7] = S_t[0] \oplus S_t[6] \end{cases}$$

See Fig. 1 for the corresponding illustration.

The Mode of Operation. Rocca is composed of four phases: initialization, processing the associated data, encryption, and finalization. The encryption algorithm of Rocca takes the following inputs: a 256-bit key $K = K_0 \parallel K_1 \in \mathbb{F}_2^{128} \times \mathbb{F}_2^{128}$, a 128-bit nonce $N \in \mathbb{F}_2^{128}$, associated data A, and a message M. The output consists of the corresponding ciphertext C and a 128-bit tag $T \in \mathbb{F}_2^{128}$. We write $\text{Enc}_K(N, A, M) = (C, T)$. The decryption of Rocca takes (K, N, A, C, T) and returns M such that $\text{Enc}_K(N, A, M) = (C, T)$, or $\perp$ indicating rejection. We write $\text{Dec}_K(N, A, C, T) = M$ or $\text{Dec}_K(N, A, C, T) = \perp$. If M is irrelevant in $\text{Dec}_K(N, A, C, T) = M$, we may write this as $\text{Dec}_K(N, A, C, T) = \top$, indicating acceptance.

To process a string X of any bit length, we define $\overline{X} = X \| 0^l$, where l represents the minimum non-negative integer required to make the length of $\overline{X}$ a multiple of 256. Additionally, if a string X has a length that is already a multiple of 256, we express it as $X = X_0 \| X_1 \| \cdots \| X_{\frac{|X|}{128}-1}$, where each X_i has a length of 128 bits.

Initialization. A 128-bit nonce N and a 256-bit key $K_0 \| K_1$ are loaded into the state S in the following way:

$$(S[0], \ldots, S[7]) \leftarrow (K_1, N, Z_0, Z_1, N \oplus K_1, 0^{128}, K_0, 0^{128})$$

Note that Z_0 and Z_1 are defined in Eq.(1). Then, 20 iterations of the round function $R(S, Z_0, Z_1)$ is applied to the state S. We call the internal state after the initialization an *initial state*.

Processing Associated Data. Associated data A is padded to $\overline{A}$ and is parsed as $\overline{A} = A_0 \| A_1 \| \cdots \| A_{d-1}$ for $d = |\overline{A}|/128$. Note that $|\overline{A}|$ is a multiple of 256 and $|A_i| = 128$ for $0 \leq i \leq d-1$. Then the state is updated as follows:

```
for $i = 0$ to $d/2 - 1$
    $R(S, A_{2i}, A_{2i+1})$
end for
```

Note that this phase is skipped if A is empty.

Processing Message. On encryption, we process a message as follows: The message M is first padded to $\overline{M}$ and is parsed as $\overline{M} = M_0 \| M_1 \| \cdots \| M_{m-1}$ for $m = |\overline{M}|/128$. Then, $\overline{M}$ is absorbed with the round function, and the corresponding ciphertext C is generated. A detailed procedure is shown as follows:

```
for $i = 0$ to $m/2 - 1$
    $C_{2i} = \mathcal{A}(S_i[1]) \oplus S_i[5] \oplus M_{2i}$
    $C_{2i+1} = \mathcal{A}(S_i[0] \oplus S_i[4]) \oplus S_i[2] \oplus M_{2i+1}$
    $R(S, M_{2i}, M_{2i+1})$
end for
```

Then, we let C be the first $|M|$ bits of $C_0 \| C_1 \| \cdots \| C_{m-1}$. Note that this phase is skipped if M is empty.

Processing Ciphertext. On decryption, the ciphertext C is first padded to $\overline{C}$, and then parsed as $\overline{C} = C_0 \| C_1 \| \cdots \| C_{m-1}$. Then, $\overline{C}$ is absorbed with the round function to generate the corresponding message M. The procedure is as follows:

```
for $i = 0$ to $m/2 - 1$
    $M_{2i} = \mathcal{A}(S_i[1]) \oplus S_i[5] \oplus C_{2i}$
    $M_{2i+1} = \mathcal{A}(S_i[0] \oplus S_i[4]) \oplus S_i[2] \oplus C_{2i+1}$
    $R(S, M_{2i}, M_{2i+1})$
end for
```

We let M be the first $|C|$ bits of $M_0 \| M_1 \| \cdots \| M_{m-1}$. Note that this phase is skipped if C is empty.

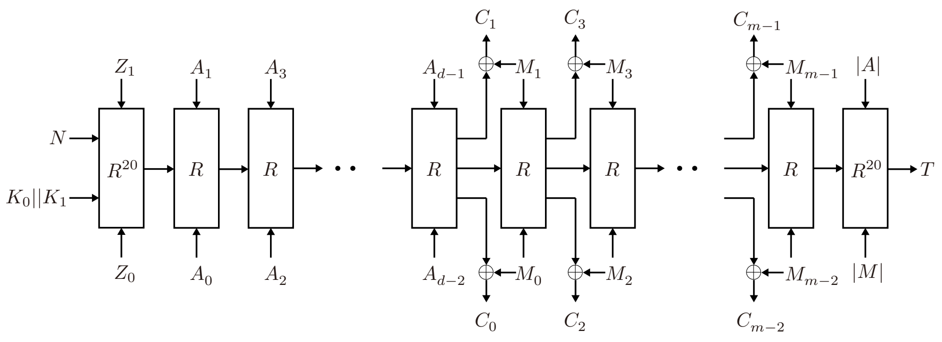

Figure 2: The encryption procedure of Rocca-v1

Finalization. After processing the message/ciphertext, the state S passes through 20 iterations of the round function $R(S, |A|, |M|)$ and then the tag is computed in the following way:

$$T = \bigoplus_{0 \le i \le 7} S[i]$$

On decryption, the computed tag is compared with the tag given as input, and M is returned if they are equal. Figure 2 shows the encryption procedure of Rocca.

2.2 Specification of Other Versions

Rocca-v2 [HII+22], Rocca-v3 [SLN+22a], and Rocca-v4 [SLN+22b] revise the specification of the initialization and finalization to counter the attack by Hosoyamada et al. in [HII+22].

Initialization. Starting from K and N, after 20 iterations of the round function, two 128-bit keys are XORed into the state S. In Rocca-v2 [HII+22], the keys are XORed as

$$S[5] \leftarrow S[5] \oplus K_0 \text{ and } S[6] \leftarrow S[6] \oplus K_1\,,$$

and in Rocca-v3 [SLN+22a] and in Rocca-v4 [SLN+22b], the keys are XORed as

$$S[0] \leftarrow S[0] \oplus K_0 \text{ and } S[4] \leftarrow S[4] \oplus K_1\,.$$

We write this process as $\mathrm{Init}(N, K_0, K_1) = S$.

Finalization. Starting from the state S, before 20 iterations of the round function, two 128-bit keys are XORed into the state S in the following way:

$$\begin{cases} S[1] \leftarrow S[1] \oplus K_0 \text{ and } S[2] \leftarrow S[2] \oplus K_1 \text{ in Rocca-v2 [HII+22]}\,, \\ S[0] \leftarrow S[0] \oplus K_0 \text{ and } S[4] \leftarrow S[4] \oplus K_1 \text{ in Rocca-v3 [SLN+22a]}\,. \end{cases}$$

Then, the round function is iterated 20 times to compute a tag T. We write this process as $\mathrm{Fin}^*(S, K) = T$. Note that Rocca-v4 does not have the key XORing at the beginning of the finalization.

Figure 3 shows the encryption procedure of Rocca-v2 and Rocca-v3.

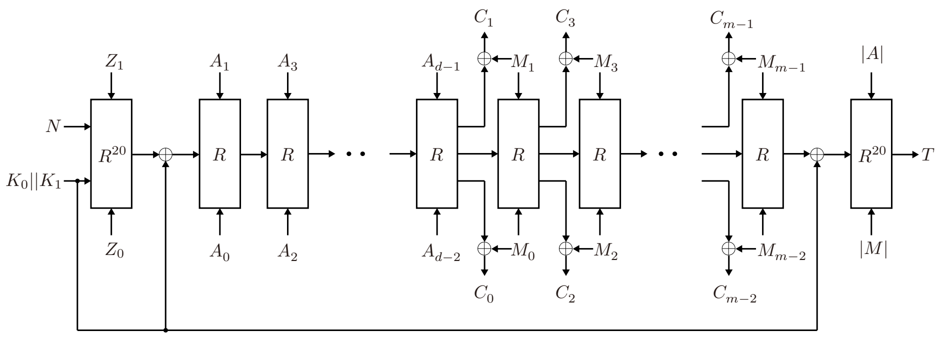

Figure 3: The encryption procedure of Rocca-v2 and Rocca-v3.

2.3 State-Recovery Attack Shown in [HII+22]

Hosoyamada et al. showed a state-recovery attack against Rocca-v1 [HII+22].

The attack exploits two plaintexts/ciphertexts using the same nonce, where the two texts have a specific difference. In the nonce-misuse setting, collecting such texts is easy by using the encryption oracle. On the other hand, in the nonce-respecting setting, it is prohibited to make two queries with the same nonce to the encryption oracle. Instead, Hosoyamada et al. exploited the decryption oracle. Since the tag size of Rocca is 128 bits and the security level is 256 bits, it is possible to collect such texts by using 2^{128} decryption queries.

We now have two texts with a common internal state. The goal is to recover the internal state. We absorb different messages whose relationship has a specific difference. The difference is diffused to the whole internal state via several rounds, but their diffused differences are known until several rounds. As a result, we have several active $\mathcal{A}$ functions whose input and output differences are known. Since the function $\mathcal{A}$ is the AES round function, we can reduce the candidate of input/output values of the active S-boxes into two or four when the input/output differences are known. By exploiting the relationship among five such active $\mathcal{A}$ functions and the meet-in-the-middle technique, Hosoyamada et al. showed the procedure to recover the whole internal state with a complexity of 2^{20}. Refer to [HII+22] for more details.

3 Key-Recovery Attacks against Rocca-v2 and Rocca-v3

In this section, we propose key-recovery attacks against Rocca-v2 [HII+22] and Rocca-v3 [SLN+22a]. We cover both nonce-misuse and nonce-respecting settings. The attacker is given the encryption and/or decryption oracles, and the goal is to recover the 256-bit secret key $K = K_0 \parallel K_1$. In [SLN+22a] and [HII+22], the position to XOR the key in the finalization phase differs. We show that both key positions allow key-recovery attacks.

We first consider Rocca-v3 [SLN+22a] in Sect. 3.1 since the attack is simpler, and Rocca-v2 [HII+22] in Sect. 3.2. Our attacks use the state-recovery in [HII+22] by Hosoyamada et al. as a subroutine, which is outlined in Sect. 2.3.

Notation. To describe our attacks, we introduce further notation. For state $S_t = (S_t[0], \ldots, S_t[7])$, we write $S_t[a..b]$ to denote $S_t[a], S_t[a+1], \ldots, S_t[b]$, and $S_t[a, b..c]$ to denote $S_t[a], S_t[b], S_t[b{+}1], \ldots, S_t[c]$. $\mathrm{Fin}_K^{\mathrm{Enc}}(S_t)$ denotes a tag T that is obtained by making an encryption query (N, A, M) such that the internal state just before the finalization phase is S_t. This corresponds to an online computation. Similarly, $\mathrm{Fin}_K^{\mathrm{Dec}}(S_t, T)$ denotes $\top$ or $\perp$ that is obtained by making a decryption query (N, A, C, T) such that the internal state just before the finalization phase is S_t. It returns $\top$ if the decryption is successful, or $\perp$ if the

Algorithm 1 Nonce-misuse key-recovery attack against Rocca-v3

```
Input: $N$                                              ▷ choose arbitrary $N$
Output: $K_0, K_1$
 1: $S_0 \leftarrow \mathrm{StateRecovery}(N)$          ▷ run the state-recovery attack on $N$
 2: for $M_0$ in range $\{0,1\}^{128}$ do
 3:     $S_1 \leftarrow R(S_0, M_0, 0^{128})$           ▷ offline computation
 4:     $T \leftarrow \mathrm{Fin}_K^{\mathrm{Enc}}(S_1)$   ▷ encryption query
 5:     $\mathrm{TagList}[T] \leftarrow M_0$            ▷ store $(T, M_0)$ to TagList
 6: end for
 7: $S_1 \leftarrow R(S_0, M_0, 0^{128})$ for $M_0 = 0^{128}$   ▷ offline computation
 8: for $k_1$ in range $\{0,1\}^{128}$ do
 9:     $T \leftarrow \mathrm{Fin}^*(S_1, K_0, k_1)$ for $K_0 = 0^{128}$   ▷ offline computation
10:     if $T$ in TagList then
11:         $M_0 \leftarrow \mathrm{TagList}[T]$        ▷ retrieve the corresponding $M_0$
12:         if $\mathrm{Init}(N, M_0, k_1) = S_0$ then
13:             return $(K_0, K_1) \leftarrow (M_0, k_1)$
14:         end if
15:     end if
16: end for
```

decryption fails or is invalid. This corresponds to an online computation. $\mathrm{Fin}^*(S_t, K_0, K_1)$ denotes the offline computation of a tag T that uses K_0, K_1, and the internal state S_t, which is the state just before the finalization phase. Finally, $\mathrm{Init}(N, K_0, K_1)$ denotes offline computation of the internal state after the initialization phase from K_0, K_1, and N.

3.1 Key-Recovery Attack against Rocca-v3

In the finalization phase of Rocca-v3 [SLN+22a], the key is XORed into the same positions as the input blocks of the previous round. Our attack makes use of this fact.

3.1.1 Case of Nonce-Misuse Setting

We first consider the nonce-misuse setting, and our key-recovery attack is presented in Algorithm 1. Following the notation in Sect. 2.1, we first establish the relationship between the queries and offline computations. In this version, the key is XORed into $S_t[0]$ and $S_t[4]$ as $S_t[0] \oplus K_0$ and $S_t[4] \oplus K_1$. Therefore, for any differential inputs $X_0, X_1 \in \{0,1\}^{128}$ to the state of the input of the finalization S_t, the following relationship holds true:

$$\mathrm{Fin}_K^{\mathrm{Enc}}(S_t[0] \oplus X_0, S_t[1..3], S_t[4] \oplus X_1, S_t[5..7]) = \mathrm{Fin}^*(S_t, K_0 \oplus X_0, K_1 \oplus X_1) \quad (2)$$

To see this, on both sides of Eq.(2), the state after XORing the key in the finalization phase becomes $(S_t[0] \oplus X_0 \oplus K_0, S_t[1..3], S_t[4] \oplus X_1 \oplus K_1, S_t[5..7])$, resulting in matching output tags. Since the inputs are XORed at the same positions as the key, we use M_0 as X_0 and M_1 as X_1. See Fig. 4 for a figure describing the case $t = 1$.

Now, consider the case $|M| = 2n$ (i.e., $t = 1$), and let M_1 be any fixed value. For simplicity, let $M_1 = 0^{128}$. If $K_0 \oplus M_0 = 0^{128}$ holds true (0^{128} can be other constant, but we consider the case $K_0 \oplus M_0 = 0^{128}$ for simplicity), then we have

$$\mathrm{Fin}_K^{\mathrm{Enc}}(S_1[0] \oplus M_0, S_1[1..3], S_1[4], S_1[5..7]) = \mathrm{Fin}^*(S_1, 0^{128}, K_1)\,. \quad (3)$$

Observe that there exists M_0 that satisfies $K_0 \oplus M_0 = 0^{128}$, and we can recover K_1 through the exhaustive search over K_1. Our attack searches for M_0 and K_1 separately. That is, we first run the state-recovery attack on N to obtain S_0 (line 1, Algorithm 1). Then, for each

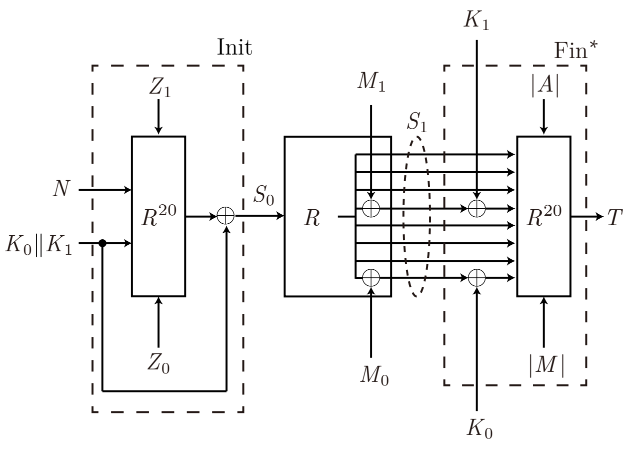

Figure 4: Illustration of the encryption process of Rocca-v3 for the case $t = 1$

$M_0 \in \{0,1\}^{128}$, we make an encryption query such that the state before the finalization is $(S_1[0] \oplus M_0, S_1[1..3], S_1[4], S_1[5..7])$. This corresponds to making an encryption query $(N, A, M_0 \parallel M_1)$ with $|A| = 0$ and $M_1 = 0^{128}$. We then obtain T, and store (T, M_0) in TagList, sorted in T (line 2–5, Algorithm 1). Then, we compute $\text{Fin}^*(S_1, 0^{128}, k_1)$ in offline for each $k_1 \in \{0,1\}^{128}$ for the exhaustive search on K_1 (line 8–16, Algorithm 1). If the same tag exists in TagList, we compute the initial state with $(K_0, K_1) = (M_0, k_1)$, and if the state becomes S_0 obtained in the state-recovery attack in line 1, (M_0, k_1) is returned as the secret key.

The attack makes 2^{128} encryption queries and makes 2^{128} offline computation through Fin* with a search over TagList, which is assumed to be run in constant time, and the memory complexity of 2^{128}, which is the size of TagList. Overall, the complexity of the attack is 2^{128}. The attack succeeds with an overwhelming probability, since we exhaustively search over all M_0 and k_1, ensuring that there always exists a combination such that $M_0 = K_0$ and $k_1 = K_1$. We also see that the state after initialization is different in most cases for different keys, and hence the verification with Init ensures that the correct key is returned.

3.1.2 Case of Nonce-Respecting Setting

In the nonce-respecting setting, extra complexity is needed for decryption queries. Algorithm 2 is the key-recovery attack in the nonce-respecting setting.

In Eq.(3), we let $M_0 = m_0 \parallel m_0'$ and $K_0 = k_0 \parallel k_0'$, where $|m_0| = |m_0'| = |k_0| = |k_0'| = 64$. Here, let $m_0' = 0^{64}, M_1 = 0^{128}$, and if $k_0 \oplus m_0 = 0^{64}$ holds true, then we have

$$\text{Fin}_K^{\text{Enc}}(S_t[0] \oplus (x_0 \parallel 0^{64}), S_1[1..3], S_1[4], S_1[5..7]) = \text{Fin}^*(S_1, 0^{64} \parallel k_0', K_1)\,.$$

Our attack searches for m_0 and (k_0', K_1) separately. Following the procedure shown in Sect. 3.1.1, we utilize 2^{128} decryption queries instead of an encryption query. This is in line 5 of Algorithm 2, and the decryption query is (N, A, C, T) with $|A| = 0$ and C being the ciphertext obtained from the state S_0 and $m_0 \parallel 0^{64} \parallel 0^{128}$ as the message, which can be computed in offline. Then, there exists m_0 that satisfies $k_0 \oplus m_0 = 0^{64}$, and we can recover k_0' and K_1 through exhaustive search.

The attack makes 2^{64+128} decryption queries and makes 2^{192} offline computation through Fin* with a search over TagList, whose size is 2^{64}. Overall, the complexity of the attack is 2^{192}, and it succeeds with an overwhelming probability.

Algorithm 2 Nonce-respecting key-recovery attack against Rocca-v3

```
Input: $N$                                                  ▷ choose arbitrary $N$
Output: $K_0, K_1$
 1: $S_0 \leftarrow \mathrm{StateRecovery}(N)$              ▷ run the state recovery attack on $N$
 2: for $m_0$ in range $\{0,1\}^{64}$ do
 3:   $S_1 \leftarrow R(S_0, m_0 \parallel 0^{64}, 0^{128})$   ▷ offline computation
 4:   for $T$ in range $\{0,1\}^{128}$ do
 5:     if $\mathrm{Fin}_K^{\mathrm{Dec}}(S_1, T) = \top$ then
 6:       $\mathrm{TagList}[T] \leftarrow m_0$              ▷ store $(T, m_0)$ to list
 7:       break
 8:     end if
 9:   end for
10: end for
11: $S_1 \leftarrow R(S_0, m_0 \parallel 0^{64}, 0^{128})$ for $m_0 = 0^{64}$
12: for $k_1$ in range $\{0,1\}^{128}$ do
13:   for $k_0'$ in range $\{0,1\}^{64}$ do
14:     $T \leftarrow \mathrm{Fin}^*(S_1, k_0 \parallel k_0', k_1)$ for $k_0 = 0^{64}$   ▷ offline computation
15:     if $T$ in TagList then
16:       $m_0 \leftarrow \mathrm{TagList}[T]$              ▷ retrieve the corresponding $m_0$
17:       if $\mathrm{Init}(N, m_0 \parallel k_0', k_1) = S_0$ then
18:         return $(K_0, K_1) \leftarrow (m_0 \parallel k_0', k_1)$
19:       end if
20:     end if
21:   end for
22: end for
```

3.2 Key-Recovery Attack against Rocca-v2

In the finalization phase of Rocca-v2 [HII+22], the key is XORed as $S_t[1] \oplus K_0, S_t[2] \oplus K_1$. For any $X_0, X_1 \in \{0,1\}^{128}$, we have

$$\mathrm{Fin}_K^{\mathrm{Enc}}(S_t[0], S_t[1] \oplus X_0, S_t[2] \oplus X_1, S_t[3..7]) = \mathrm{Fin}^*(S_t, K_0 \oplus X_0, K_1 \oplus X_1)\,.$$

Now, let X be a variable over some set $\mathcal{X}$, and let $f, g : \mathcal{X} \to \{0,1\}^{128}$ be functions. Assume that we can express X_0 and X_1 as $X_0 = f(X)$ and $X_1 = g(X)$. If $K_0 \oplus X_0 = 0^{128}$ and $k_1 = K_1 \oplus g(X)$ hold true, then we have

$$\mathrm{Fin}_K^{\mathrm{Enc}}(S_t[0], S_t[1] \oplus f(X), S_t[2] \oplus g(X), S_t[3..7]) = \mathrm{Fin}^*(S_t, 0^{128}, k_1)\,.$$

Assuming that computing $f(X)$ for 2^{128} different values of X covers most of the space of $\{0,1\}^{128}$, performing an exhaustive search for both X and k_1 guarantees the existence of X and k_1 such that $K_0 \oplus f(X) = 0^{128}$ and $k_1 = K_1 \oplus g(X)$ with a high probability. By obtaining X and k_1, we succeed in the key-recovery attack by using the same procedure as in Sect. 3.1.

We consider the case $|M| = 8n$ (i.e., $t = 4$). The approach of the attack is to develop a method to compute the inputs $M = (M_0, \ldots, M_7)$ for 4 rounds, where $f(M)$ and $g(M)$ are XORed into $S_4[1]$ and $S_4[2]$, respectively, while keeping the values of other state blocks $S_4[0, 3..7]$ unchanged, in order to make $\mathrm{Fin}_K^{\mathrm{Enc}}$ queries.

The internal states of S_1 to S_4 for the last 3 rounds out of the 4 rounds are represented as follows:

$$
\begin{aligned}
&S_4[0] = S_1[7] \oplus M_2 \oplus \mathcal{A}(S_1[5]) \oplus S_1[4] \oplus M_6 \\
&S_4[1] = \mathcal{A}(S_1[0] \oplus S_1[6] \oplus M_4) \oplus S_1[7] \oplus M_2 \oplus \mathcal{A}(S_1[5]) \oplus S_1[4] \\
&S_4[2] = \mathcal{A}(S_1[7] \oplus M_2) \oplus S_1[0] \oplus S_1[6] \oplus \mathcal{A}(\mathcal{A}(S_1[4]) \oplus S_1[3]) \oplus S_1[3] \oplus M_3 \\
&S_4[3] = \mathcal{A}(\mathcal{A}(S_1[0]) \oplus S_1[7] \oplus \mathcal{A}(S_1[5]) \oplus S_1[4]) \oplus \mathcal{A}(S_1[7] \oplus M_2) \oplus S_1[0] \oplus S_1[6] \\
&S_4[4] = \mathcal{A}(S_1[1] \oplus S_1[6]) \oplus \mathcal{A}(S_1[0]) \oplus S_1[7] \oplus M_7 \\
&S_4[5] = \mathcal{A}(\mathcal{A}(S_1[2]) \oplus S_1[1] \oplus M_5) \oplus \mathcal{A}(S_1[1] \oplus S_1[6]) \oplus \mathcal{A}(S_1[0]) \oplus S_1[7] \\
&S_4[6] = \mathcal{A}(\mathcal{A}(S_1[3] \oplus M_3) \oplus \mathcal{A}(S_1[2]) \oplus S_1[1]) \oplus \mathcal{A}(S_1[2]) \oplus S_1[1] \oplus M_5 \\
&S_4[7] = S_1[0] \oplus S_1[6] \oplus M_4 \oplus \mathcal{A}(\mathcal{A}(S_1[4]) \oplus S_1[3]) \oplus S_1[3] \oplus M_3
\end{aligned} \tag{4}
$$

The input message blocks are highlighted in red. From the above equations, for any S_1, we can find inputs $M_2, \ldots, M_7$ that fix the values of $S_4[0, 3..7]$ to any desired value in constant time. In more detail, we proceed as follows:

- We first fix M_2 so that $S_4[3]$ becomes a desired value.
- We then fix M_6 so that $S_4[0]$ becomes a desired value.
- We then fix M_5 so that $S_4[5]$ becomes a desired value.
- We then fix M_3 so that $S_4[6]$ becomes a desired value.
- We then fix M_4 so that $S_4[7]$ becomes a desired value.
- Finally, we fix M_7 so that $S_4[4]$ becomes a desired value.

Therefore, for any S_0, when we modify the values of M_0 and M_1, the values of the other 6 blocks except for $S_4[1]$ and $S_4[2]$ can be kept unchanged by choosing suitable $M_2, \ldots, M_7$, and the values of $S_4[1]$ and $S_4[2]$ depend on M_0 and M_1. The above steps give us a way to find inputs $M_2, \ldots, M_7$ that satisfy this condition.

From the above, we can express $X_0 = f(M_0, M_1)$ and $X_1 = g(M_0, M_1)$, and by performing a search over M_0 and M_1 with a complexity of 2^{128}, we can perform an exhaustive search for $f(M_0, M_1)$. Assuming that f behaves as a random function, the probability that we have M satisfying $K_0 \oplus f(M_0, M_1) = 0^{128}$ by 2^{128} searches over M_0 and M_1 is estimated as $(1 - e^{-1})$ from Poisson distribution. For this M, since we exhaustively search over k_1, there exists k_1 that satisfies $k_1 = K_1 \oplus g(M_0, M_1)$, allowing us to determine K_0 and K_1.

We consider the case $S_4 = 0^{1024}$, and our key-recovery attack in the nonce-misuse setting is presented in Algorithm 3. We can perform a key-recovery attack with a complexity of 2^{128}. Our key-recovery attack in the nonce-respecting setting is presented in Algorithm 4. The time complexity of the attack is 2^{192}.

We remark that [DFI+24] uses a similar approach to compute $M_2, \ldots, M_7$ in their committing attack, citing [TI23] as a reference.

4 Rocca with Stronger Finalization

The key-recovery attacks in the previous section exploit the key XORing of the finalization. To counter the attack, one option is to remove the key XORing, as in Rocca-v4 [SLN+22b]. However, it means a practical threat of universal forgery in the nonce-misuse setting. Therefore, a more robust countermeasure is to use a stronger finalization phase. In this section, we consider a new version of Rocca, where an arbitrary linear expansion of the key is XORed into the state before the last 20 iterations of the round function. Then, the above

Algorithm 3 Nonce-misuse key-recovery attack against Rocca-v2

```
Input: $N$                                                          ▷ choose arbitrary $N$
Output: $K_0, K_1$
 1: $S_0 \leftarrow \mathrm{StateRecovery}(N)$                        ▷ run the state-recovery attack on $N$
 2: for $M_0$ in range $\{0,1\}^{128}$ do
 3:     $S_1 \leftarrow R(S_0, M_0, M_1)$ for $M_1 = 0^{128}$                ▷ offline computation
 4:     $M_2 \leftarrow S_1[7] \oplus \mathcal{A}^{-1}(\mathcal{A}(\mathcal{A}(S_1[0]) \oplus S_1[7] \oplus \mathcal{A}(S_1[5]) \oplus S_1[4]) \oplus S_1[0] \oplus S_1[6])$
 5:     $M_6 \leftarrow S_1[7] \oplus M_2 \oplus \mathcal{A}(S_1[5]) \oplus S_1[4]$
 6:     $M_5 \leftarrow \mathcal{A}(S_1[2]) \oplus S_1[1] \oplus \mathcal{A}^{-1}(\mathcal{A}(S_1[1] \oplus S_1[6]) \oplus \mathcal{A}(S_1[0]) \oplus S_1[7])$
 7:     $M_3 \leftarrow S_1[3] \oplus \mathcal{A}^{-1}(\mathcal{A}(S_1[2]) \oplus S_1[1] \oplus \mathcal{A}^{-1}(\mathcal{A}(S_1[2]) \oplus S_1[1] \oplus M_5))$
 8:     $M_4 \leftarrow S_1[0] \oplus S_1[6] \oplus \mathcal{A}(\mathcal{A}(S_1[4]) \oplus S_1[3]) \oplus S_1[3] \oplus M_3$
 9:     $M_7 \leftarrow \mathcal{A}(S_1[1] \oplus S_1[6]) \oplus \mathcal{A}(S_1[0]) \oplus S_1[7]$
10:     $S_4 \leftarrow R(R(R(S_1, M_2, M_3), M_4, M_5), M_6, M_7)$            ▷ $S_4[0, 3..7] = 0^{128 \cdot 6}$
11:     $T \leftarrow \mathrm{Fin}^{\mathrm{Enc}}_K(S_4)$                               ▷ encryption query
12:     $\mathrm{TagList}[T] \leftarrow S_4[1], S_4[2]$                 ▷ store $(T, S_4[1], S_4[2])$ to TagList
13: end for
14: for $k_1$ in range $\{0,1\}^{128}$ do
15:     $T \leftarrow \mathrm{Fin}^*(S_4, K_0, k_1)$ for $S_4 = 0^{1024}, K_0 = 0^{128}$        ▷ offline computation
16:     if $T$ in TagList then
17:         $S_4[1], S_4[2] \leftarrow \mathrm{TagList}[T]$          ▷ retrieve the corresponding $S_4[1], S_4[2]$
18:         if $\mathrm{Init}(N, S_4[1], S_4[2] \oplus k_1) = S_0$ then
19:             return $(K_0, K_1) \leftarrow (S_4[1], S_4[2] \oplus k_1)$
20:         end if
21:     end if
22: end for
```

attack, which adjusts each block step-by-step, is no longer available. It turns out that this still allows a key-recovery attack faster than the exhaustive key search. In order to show this, we first establish an algorithm to derive a message that interpolates two internal states. We then show the key-recovery attack using the state-interpolation algorithm. We also discuss possible countermeasures to strengthen the finalization phase further. Using the ideally secure keyed finalization is an option to avoid the key-recovery attack. Although it still allows non-trivial universal forgery attacks, the required complexity is not practical.

We start by proposing our state-interpolation algorithm in Sect. 4.1, followed by a key-recovery attack against Rocca with an arbitrary linear key expansion and possible countermeasures in Sect. 4.2. We finally present a universal forgery attack against Rocca with the ideal keyed finalization in Sect. 4.3.

4.1 Deriving Messages to Interpolate Internal States

In this section, we propose an algorithm to derive a message $M = (M_0, \ldots, M_7)$ of 8 blocks that interpolates two internal states (S_0, S_4). That is, for any given S_0 and S_4, the state-interpolation algorithm returns $M = (M_0, \ldots, M_7)$ such that

$$S_4 = R(R(R(R(S_0, M_0, M_1), M_2, M_3), M_4, M_5), M_6, M_7)\,,$$

where the time complexity of the algorithm is 2^{160}.

Input Relationships. Figure 5 shows the internal state of Rocca for the state-interpolation algorithm. In the figure, the colors have the following semantics:

- Red-colored blocks are given values.

Algorithm 4 Nonce-respecting key-recovery attack against Rocca-v2

```
Input: $N$                                                        ▷ choose arbitrary $N$
Output: $K_0, K_1$
 1: $S_0 \leftarrow \mathrm{StateRecovery}(N)$                     ▷ run the state recovery attack on $N$
 2: for $m_0$ in range $\{0,1\}^{128}$ do
 3:     $S_1 \leftarrow R(S_0, m_0 \parallel m_0', M_1)$ for $m_0' = 0^{64}, M_1 = 0^{128}$     ▷ offline computation
 4:     $M_2 \leftarrow S_1[7] \oplus \mathcal{A}^{-1}(\mathcal{A}(\mathcal{A}(S_1[0]) \oplus S_1[7] \oplus \mathcal{A}(S_1[5]) \oplus S_1[4]) \oplus S_1[0] \oplus S_1[6])$
 5:     $M_6 \leftarrow S_1[7] \oplus M_2 \oplus \mathcal{A}(S_1[5]) \oplus S_1[4]$
 6:     $M_5 \leftarrow \mathcal{A}(S_1[2]) \oplus S_1[1] \oplus \mathcal{A}^{-1}(\mathcal{A}(S_1[1] \oplus S_1[6]) \oplus \mathcal{A}(S_1[0]) \oplus S_1[7])$
 7:     $M_3 \leftarrow S_1[3] \oplus \mathcal{A}^{-1}(\mathcal{A}(S_1[2]) \oplus S_1[1] \oplus \mathcal{A}^{-1}(\mathcal{A}(S_1[2]) \oplus S_1[1] \oplus M_5))$
 8:     $M_4 \leftarrow S_1[0] \oplus S_1[6] \oplus \mathcal{A}(\mathcal{A}(S_1[4]) \oplus S_1[3]) \oplus S_1[3] \oplus M_3$
 9:     $M_7 \leftarrow \mathcal{A}(S_1[1] \oplus S_1[6]) \oplus \mathcal{A}(S_1[0]) \oplus S_1[7]$
10:     $S_4 \leftarrow R(R(R(S_1, M_2, M_3), M_4, M_5), M_6, M_7)$     ▷ $S_4[0, 3..7] = 0^{128 \cdot 6}$
11:     for $T$ in range $\{0,1\}^{128}$ do
12:         if $\mathrm{Fin}_K^{\mathrm{Dec}}(S_4, T) = \top$ then
13:             $\mathrm{TagList}[T] \leftarrow S_4[1], S_4[2]$     ▷ store $(T, S_4[1], S_4[2])$ to TagList
14:             break
15:         end if
16:     end for
17: end for
18: for $k_1$ in range $\{0,1\}^{128}$ do
19:     for $k_0'$ in range $\{0,1\}^{64}$ do
20:         $T \leftarrow \mathrm{Fin}^*(S_4, k_0 \parallel k_0', k_1)$ for $S_4 = 0^{1024}, k_0 = 0^{64}$     ▷ offline computation
21:         if $T$ in TagList then
22:             $S_4[1], S_4[2] \leftarrow \mathrm{TagList}[T]$     ▷ retrieve the corresponding $S_4[1], S_4[2]$
23:             if $\mathrm{Init}(N, S_4[1] \oplus 0^{64} \parallel k_0', S_4[2] \oplus k_1) = S_0$ then
24:                 return $(K_0, K_1) \leftarrow (S_4[1] \oplus 0^{64} \parallel k_0', S_4[2] \oplus k_1)$
25:             end if
26:         end if
27:     end for
28: end for
```

- Blue-colored blocks are computed by determining M_0.
- Green-colored blocks are computed by determining M_1.
- Yellow-colored blocks are computed by determining M_0 and M_1.

As shown in Fig. 5, considering M_0 and M_1 as variables, the internal states are calculated in the order of the arrows. If the internal states determined by M_0 and M_1 do not cause a contradiction, $M_2, \dots, M_7$ that interpolate between S_0 and S_4 can be found. $S_3[1]$ and $S_2[0]$, the states enclosed by the red dashed line in Fig. 5, are the only states that can cause a contradiction if we follow the arrows of Fig. 5. These states can be represented in two ways using M_0, M_1. Since these values must match, we identify two equations, Eqs.(5) and (6), as follows:

$$S_3[1] = S_4[3] \oplus \mathcal{A}(S_2[1] \oplus S_2[6]) = S_4[2] \oplus S_2[4] \oplus \mathcal{A}(S_2[5]) \tag{5}$$

$$S_2[0] = S_2[6] \oplus S_3[7] = \mathcal{A}^{-1}(S_2[7] \oplus S_3[1]) \tag{6}$$

State-Interpolation Algorithm. We show an algorithm that computes M_0 and M_1 such that both Eqs.(5) and (6) are satisfied. Then, the remaining message blocks $M_2, \dots, M_7$ can be efficiently computed.

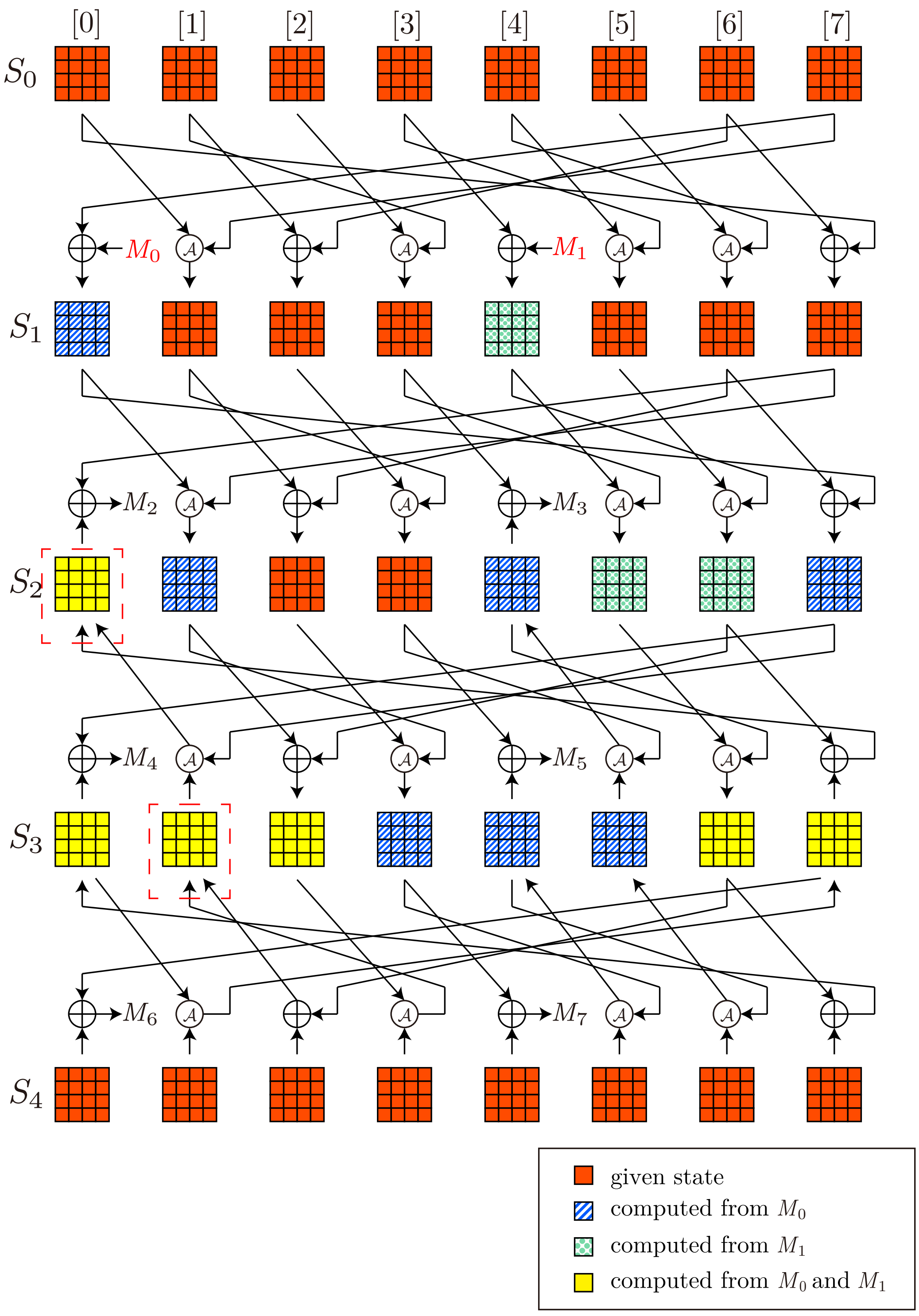

Figure 5: The internal state of Rocca. S_0 and S_4 are given. The algorithm returns $M_0, \ldots, M_7$.

1. Fix M_0 arbitrarily. Then rewrite Eq.(5) as an expression of M_1 with M_0 being fixed. In detail, first, we expand Eq.(5) with respect to M_1 as follows:

$$\begin{aligned} S_3[1] &= S_4[3] \oplus \mathcal{A}(S_2[1] \oplus \mathcal{A}(S_1[5]) \oplus S_0[3] \oplus M_1) \\ &= S_4[2] \oplus S_2[4] \oplus \mathcal{A}(\mathcal{A}(S_0[3] \oplus M_1) \oplus S_1[3]) \end{aligned} \tag{7}$$

Here, the blue-colored blocks in Fig. 5 are highlighted in blue in Eq.(7). From red-colored blocks and blue-colored blocks, we define X and Y as follows:

$$\begin{aligned} X &= S_4[3] \oplus S_4[2] \oplus S_2[4] \\ Y &= S_2[1] \oplus \mathcal{A}(S_1[5]) \end{aligned}$$

By using X and Y, Eq.(7) can be expressed as follows:

$$\mathcal{A}(M_1' \oplus Y) = \mathcal{A}(\mathcal{A}(M_1') \oplus S_1[3]) \oplus X \tag{8}$$

Here, M_1' is defined as

$$M_1' = M_1 \oplus S_0[3] = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} \end{bmatrix}.$$

Now from the linearity of MixColumns and ShiftRows, by defining X' as $X' = \mathrm{SR}^{-1} \circ \mathrm{MC}^{-1}(X)$, Eq.(8) can be written as

$$\mathrm{SB}(M_1' \oplus Y) = \mathrm{SB}(\mathrm{MC} \circ \mathrm{SR} \circ \mathrm{SB}(M_1') \oplus S_1[3]) \oplus X'. \tag{9}$$

2. Solve Eq.(9) for M_1'.

   We show the process in this step in Figs. 6(a)–(d).

   (a) We focus on the blue-colored bytes in Fig. 6(a). Then, Eq.(9) is expressed as

$$\mathrm{SB}\left(\begin{bmatrix} x_{0,0} \\ x_{1,0} \\ x_{2,0} \\ x_{3,0} \end{bmatrix} \oplus Y\right) = \mathrm{SB}\left(\mathrm{MC}\begin{bmatrix} \mathrm{Sb}(x_{0,0}) \\ \mathrm{Sb}(x_{1,1}) \\ \mathrm{Sb}(x_{2,2}) \\ \mathrm{Sb}(x_{3,3}) \end{bmatrix} \oplus S_1[3]\right) \oplus X'. \tag{10}$$

   Here, we abuse the notation to apply MC and SB on a word of 32-bit state. Now, we first fix $x_{0,0}, x_{1,1}, x_{2,2}$ on the right hand side of Eq.(10). Then since $x_{0,0}$ is fixed, the corresponding first row value of the left hand side of Eq.(10) is fixed, and from this, we obtain a unique value of $x_{3,3}$ that is consistent with the first row. Then, all the values in the left hand side have been fixed, and we have $x_{1,0}, x_{2,0}, x_{3,0}$, which are the bytes marked with a red circle in Fig. 6(a), that satisfy the equality of Eq.(10).

   (b) We focus on the blue-colored bytes in Fig. 6(b). Gray-colored bytes are fixed bytes in Step (a). We obtain the following equation:

$$\mathrm{SB}\left(\begin{bmatrix} x_{0,1} \\ x_{1,1} \\ x_{2,1} \\ x_{3,1} \end{bmatrix} \oplus Y\right) = \mathrm{SB}\left(\mathrm{MC}\begin{bmatrix} \mathrm{Sb}(x_{0,1}) \\ \mathrm{Sb}(x_{1,2}) \\ \mathrm{Sb}(x_{2,3}) \\ \mathrm{Sb}(x_{3,0}) \end{bmatrix} \oplus S_1[3]\right) \oplus X' \tag{11}$$

   We obtain the system of equations in $x_{1,2}$ and $x_{2,3}$ by trying $x_{0,1}$ as indicated by the first and the second rows of Eq.(11). For each of $(x_{0,1}, x_{1,2}, x_{2,3})$, the right hand side of Eq.(11) is fixed, implying that the values of $x_{2,1}$ and $x_{3,1}$ are fixed as well.

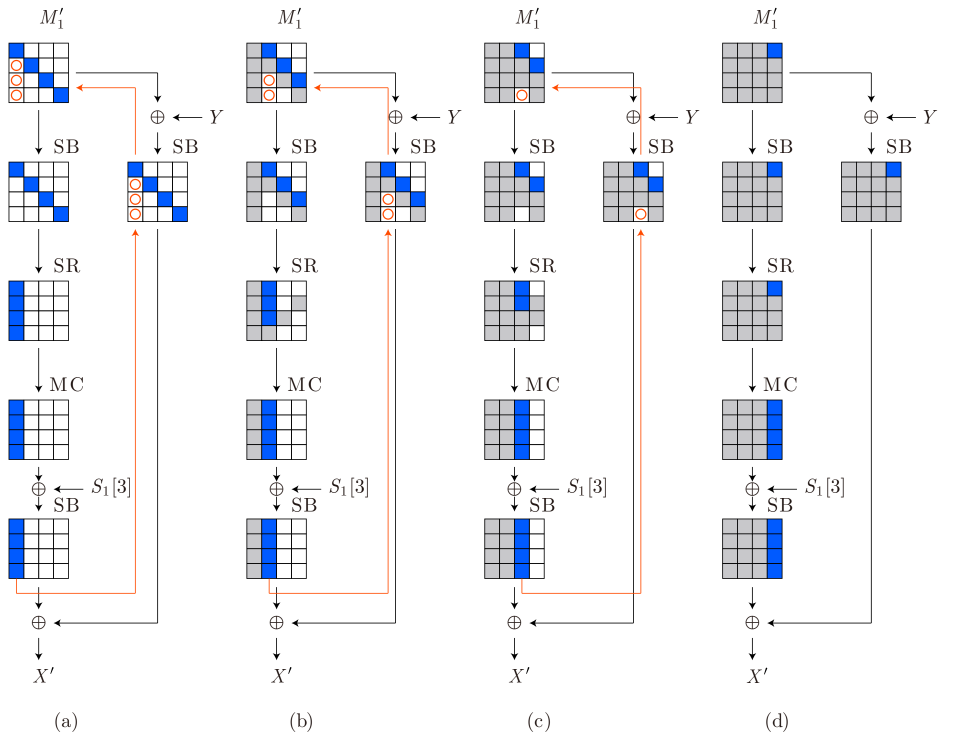

Figure 6: Illustration of Eq.(9). Our goal is to obtain M_1'.

(c) We focus on the blue-colored bytes in Fig. 6(c). We obtain

$$\mathrm{SB}\left(\begin{bmatrix} x_{0,2} \\ x_{1,2} \\ x_{2,2} \\ x_{3,2} \end{bmatrix} \oplus Y\right) = \mathrm{SB}\left(\mathrm{MC}\begin{bmatrix} \mathrm{Sb}(x_{0,2}) \\ \mathrm{Sb}(x_{1,3}) \\ \mathrm{Sb}(x_{2,0}) \\ \mathrm{Sb}(x_{3,1}) \end{bmatrix} \oplus S_1[3]\right) \oplus X'. \quad (12)$$

We then have the system of equations in $x_{0,2}$ and $x_{1,3}$ that is obtained from the first, second, and the third rows of Eq.(12). The right hand side of Eq.(12) is fixed, implying that the value of $x_{3,2}$ is fixed as well.

(d) We focus on the blue-colored byte in Fig. 6(d). We have

$$\mathrm{SB}\left(\begin{bmatrix} x_{0,3} \\ x_{1,3} \\ x_{2,3} \\ x_{3,3} \end{bmatrix} \oplus Y\right) = \mathrm{SB}\left(\mathrm{MC}\begin{bmatrix} \mathrm{Sb}(x_{0,3}) \\ \mathrm{Sb}(x_{1,0}) \\ \mathrm{Sb}(x_{2,1}) \\ \mathrm{Sb}(x_{3,2}) \end{bmatrix} \oplus S_1[3]\right) \oplus X'. \quad (13)$$

If there exists $x_{0,3}$ that satisfies the equality of Eq.(13), we have solved Eq.(9), and we obtain M_1' (and hence M_1) for Eq.(8). Otherwise, repeat Step (a) by changing $x_{0,0}, x_{1,1}, x_{2,2}, x_{0,1}$.

3. Check whether Eq.(6) holds with the obtained M_0 and M_1. If it does, proceed to Step 5, otherwise proceed to Step 4.

4. Repeat Step 1 to Step 3 by changing M_0.

5. Compute $M_2, \ldots, M_7$ from M_0 and M_1.

In Step 2, we try 4 bytes and thus the time complexity of this step is 2^{32}. The probability that Eq.(6) holds with M_0 and M_1 derived in Step 2 is expected to be 2^{-128}. Therefore, the total time complexity of our interpolation algorithm is $2^{128+32} = 2^{160}$.

We remark that it is straightforward to generalize the algorithm to interpolate two states S_0 and S_t for any $t \geq 4$. That is, we can run the above state-interpolation algorithm on (S_{t-4}, S_t), where S_{t-4} is the state after injecting any message on S_0.

4.2 Key-Recovery Attack against Rocca with Arbitrary Linear Key Expansion

In this section, we show that Rocca with an arbitrary linear key expansion still allows a key-recovery attack in the nonce-respecting setting. This in particular shows that the choice of the key position to XOR at the beginning of the finalization phase is irrelevant to the security of Rocca. We consider the case that K_0 and K_1 are XORed into the state with an arbitrary linear key expansion. That is, the state S_t is updated as $S_t \oplus L(K_0 \parallel K_1)$, where $L : \{0,1\}^{256} \to \{0,1\}^{1024}$ is a linear and injective mapping. We show that this general case still allows key-recovery attacks.

We propose the attack in Algorithm 5. Let k_0 and k_1 be variables such that $|k_0| = 48, |k'_0| = 80$ and $|k_1| = 128$. If $(K_0, K_1) = (k_0 \parallel k'_0, k_1)$, then we have

$$\begin{aligned}
\mathrm{Fin}^{\mathrm{Enc}}_K(L(k_0 \parallel 0^{208})) &= \mathrm{Fin}^*(L(k_0 \parallel 0^{208}), k_0 \parallel k'_0, k_1) \\
&= \mathrm{Fin}^*(L(k_0 \parallel 0^{208}) \oplus L(k_0 \parallel 0^{208}), 0^{48} \parallel k'_0, k_1) \\
&= \mathrm{Fin}^*(L(0^{256}), 0^{48} \parallel k'_0, k_1)\,.
\end{aligned}$$

First, we run the state-recovery attack on N to obtain S_0 (line 1, Algorithm 5). We can create an arbitrary state with a complexity of 2^{160} by using the interpolation algorithm in Sect. 4.1 (line 3, Algorithm 5). Then, we make a decryption query (N, A, C, T) (line 6, Algorithm 5), where $|A| = 0$ and C is the ciphertext of $M = (M_0, \ldots, M_7)$ obtained in line 3. Note that the state before the finalization is $S_4 = L(k_0 \parallel 0^{208})$ for each k_0. We obtain T, and store it in TagList.

Then, we compute $\mathrm{Fin}^*(L(0^{256}), 0^{48} \parallel k'_0, k_1)$ for each k'_0 and k_1 in offline (line 14, Algorithm 5). If the same tag exists in TagList, we compute the initialization phase with $(K_0, K_1) = (k_0 \parallel k'_0, k_1)$, and when the state is the same as S_0, $(k_0 \parallel k'_0, k_1)$ is returned as the key.

Since we search over k_0 and (k'_0, k_1) separately, the key-recovery in the nonce-respecting setting works with a complexity of 2^{208}.

In the nonce-misuse setting, 2^{128} decryption queries can be replaced with one encryption query. However, the overall complexity remains 2^{208} as the 2^{160} complexity of the state-interpolation algorithm in Sect. 4.1 dominates the complexity.

Possible Countermeasures. Our analyses have shown that XORing with linearly expanded key in the finalization phase does not prevent key-recovery attacks. In response, we consider several approaches of countermeasures.

First, using a non-linear key expansion is one potential countermeasure. However, it is essential to carefully choose an appropriate non-linear function, as using an unsuitable one may fail to meet the required security standards. Second, we might use keyed permutation instead of the public permutation and key XORing. When the ideal keyed permutation is used, the key-recovery attack is impossible, even if the attacker can access the keyed permutation directly.

Algorithm 5 Key-recovery attack against Rocca with arbitrary linear key expansion

```
Input: $N$                                                          ▷ choose arbitrary $N$
Output: $K_0, K_1$
 1: $S_0 \leftarrow \mathrm{StateRecovery}(N)$                         ▷ run the state-recovery attack on $N$
 2: for $k_0$ in range $\{0,1\}^{48}$ do
 3:   $(M_0, \ldots, M_7) \leftarrow \mathrm{Interpolate}(S_0, L(k_0 \parallel 0^{208}))$   ▷ $2^{160}$ time complexity
 4:   $S_4 \leftarrow R^4(S_0, M_0, \ldots, M_7)$                       ▷ $S_4 = L(k_0 \parallel 0^{208})$
 5:   for $T$ in range $\{0,1\}^{128}$ do
 6:     if $\mathrm{Fin}_K^{\mathrm{Dec}}(S_4, T) = \top$ then
 7:       $\mathrm{TagList}[T] \leftarrow k_0$
 8:       break
 9:     end if
10:   end for
11: end for
12: $S_4 \leftarrow L(0^{256})$
13: for $k_1$ in range $\{0,1\}^{128}$ do
14:   for $k'_0$ in range $\{0,1\}^{80}$ do
15:     $T \leftarrow \mathrm{Fin}^*(S_4, k_0 \parallel k'_0, k_1)$ for $k_0 = 0^{48}$
16:     if $T$ in TagList then
17:       $k_0 \leftarrow \mathrm{TagList}[T]$                            ▷ retrieve the corresponding $k_0$
18:       if $\mathrm{Init}(N, k_0 \parallel k'_0, k_1) = S_0$ then
19:         return $(K_0, K_1) \leftarrow (k_0 \parallel k'_0, k_1)$
20:       end if
21:     end if
22:   end for
23: end for
```

4.3 Universal Forgery against Rocca with Ideal Keyed Finalization

Universal forgery is a type of forgery attacks where an attacker is given a challenge (N^*, A^*, M^*) and outputs (C^*, T^*) such that $\mathrm{Enc}_K(N^*, A^*, M^*) = (C^*, T^*)$. The attacker can use encryption and decryption oracles. However, the attacker cannot make an encryption query (N^*, A^*, M^*).

4.3.1 Generic Universal Forgery and Results from [HII+22]

The generic complexity of the universal forgery against all the version of Rocca, or more generally, against any online AEAD scheme, is $2^{|T|}$. In an online AEAD scheme, for a fixed key, nonce, and associated data, the first i bits of the ciphertext depends only on the first i bits of the message. The generic attack works as follows: Let $M = M^* \parallel M'$ for some $|M'| \geq 1$. The attacker makes an encryption query (N^*, A^*, M) to receive C and T, where C^* is the first $|M^*|$ bits of C. The attacker makes $2^{|T|}$ decryption queries to obtain the correct tag T^* for (N^*, A^*, C^*), and this works in the nonce-respecting or nonce-misuse setting.

For Rocca, after a state-recovery attack of [HII+22], universal forgery can be executed with constant complexity against non-keyed finalization version of Rocca (Rocca-v1 and Rocca-v4). While the complexity of the state-recovery attack is 2^{128} in the nonce-respecting setting, it is 2^{20} in the nonce-misuse setting. Thus, the threat is practical in the nonce-misuse setting, and it motivates us to use the keyed finalization. An ideally secure keyed finalization can be an option, however, it is still unclear whether using it is promising against universal forgery. We consider Rocca using the ideally secure keyed finalization and show universal forgery in the nonce-misuse setting with a practical data complexity.

Algorithm 6 Procedure of universal forgery for the case $t > 4$

```
Input: $(N^*, A^*, M^*)$
Output: $(C^*, T^*)$
1: $S_0^* \leftarrow \mathrm{StateRecovery}(N^*)$
2: $(C^*, S_t^*) \leftarrow \mathrm{Encryption}(S_0^*, A^*, M^*)$
3: $(A', M') \leftarrow \mathrm{Interpolate}(S_0^*, S_t^*)$
4: $(C', T^*) \leftarrow \mathrm{Enc}_K(N^*, A', M')$
5: return $(C^*, T^*)$
```

Algorithm 7 Procedure of universal forgery for the case $t = 4$

```
Input: $(N^*, A^*, M^*)$
Output: $(C^*, T^*)$
1: $S_0^* \leftarrow \mathrm{StateRecovery}(N^*)$
2: $(C^*, S_t^*) \leftarrow \mathrm{Encryption}(S_0^*, A^*, M^*)$
3: fix $N' \neq N^*$ arbitrarily
4: $S_0' \leftarrow \mathrm{StateRecovery}(N')$
5: $(A', M') \leftarrow \mathrm{Interpolate}(S_0', S_t^*)$
6: $(C', T^*) \leftarrow \mathrm{Enc}_K(N', A', M')$
7: return $(C^*, T^*)$
```

We use the state-interpolation algorithm from Sect. 4.1, and hence the time complexity is impractical. Note that the attack here applies to other versions with keyed finalization.

4.3.2 Nonce-Misuse Universal Forgery with Practical Query Complexity

In the nonce-misuse setting, the above-mentioned generic universal forgery requires 2^{128} online complexity. The state-interpolation algorithm shown in Sect. 4.1 enables us to reduce the online complexity to a practical range in the nonce-misuse setting, although the offline complexity increases instead. As a limitation, it requires at least 8 blocks of associated data and message, i.e., $t \geq 4$, where $|\overline{A^*}| + |\overline{M^*}| = 256t$. The procedure is slightly different for $t > 4$ and $t = 4$. We first consider the case $t > 4$, followed by the case $t = 4$.

Case $t > 4$. We present the universal forgery in Algorithm 6. We first run StateRecovery in [HII+22] to recover the initial state S_0^* for the challenge nonce N^*. We next run $\mathrm{Encryption}(S_0^*, A^*, M^*)$ to output the ciphertext C^* and the state S_t^* as the offline computation. Then, we run the state-interpolation algorithm from Sect. 4.1 on S_0^* and S_t^* to obtain A' and M' that interpolate the states. Here, we use (A', M') such that $(A', M') \neq (A^*, M^*)$, $|A'| = |A^*|$, and $|M'| = |M^*|$. We finally make an encryption query (N^*, A', M') to receive (C', T^*), where the tag becomes the one for the challenge.

Case $t = 4$. In this case, (A^*, M^*) may be the unique input blocks that interpolate S_0^* and S_4^*. We therefore use a different nonce than the nonce in the challenge.

We present the procedure in Algorithm 7. First, we run the state-recovery attack and obtain S_0^* for N^*. Second, we compute C^* and S_t^* from (S_0^*, A^*, M^*) in offline. Next, we fix any nonce $N' \neq N^*$, and run the state-recovery attack to obtain S_0'. Then, we compute (A', M') that interpolates S_0' and S_t^* with the interpolation algorithm from Sect. 4.1, where $|A'| = |A^*|$ and $|M'| = |M^*|$. Finally, we make an encryption query (N', A', M') to the encryption oracle and obtain (C', T^*).

Attacks above require only three and five encryption queries for $t > 4$ and $t =$

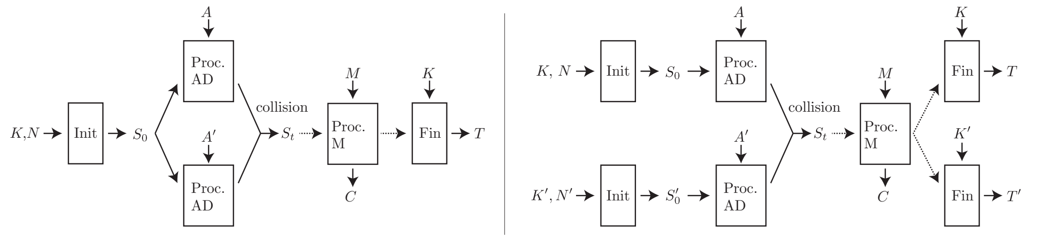

Figure 7: Two CMT attacks against several versions of Rocca. The left figure shows the CMT-3 attack. The right figure shows the CMT-2 attack, which can be converted into CMT-1 or FROB when the finalization does not involve the key.

4, respectively. On the other hand, the offline time complexity is 2^{160} for the state-interpolation algorithm.

5 Committing Security

Committing security has recently been actively discussed as a new demanded security feature for AEAD schemes. There are several kinds of committing security notions, and we discuss CMT-3, CMT-2, CMT-1 [BH22], and FROB [FOR17] security in this paper.

CMT-ℓ security requires the infeasibility of an attacker to determine (K, N, A, M) and (K', N', A', M') such that $\mathrm{Enc}_K(N, A, M) = \mathrm{Enc}_{K'}(N', A', M')$ with the condition that the tuple of the first ℓ inputs of the AEAD differs. In other words, it commits to the first ℓ inputs. FROB security is a stronger security notion than CMT-1, which requires $N = N'$ in addition to the condition for CMT-1.

We present two types of CMT attacks. The first attack breaks the CMT-3 security, where identical key and nonce are used. Therefore, it has the identical output of the initialization. We then show a pair of (distinct) associated data such that the internal state collides after several rounds. The second attack breaks the CMT-2 security, where the output of the initialization takes a different state because it uses a different key and nonce. We then show a pair of associated data such that the internal state collides after several rounds. This attack is trivially converted to the CMT-1/FROB attack when the finalization does not involve the secret key like Rocca-v1 or Rocca-v4. Figure 7 shows the high-level overview of these attacks.

5.1 CMT-3 Attack

Given the identical initial state after the initialization, the CMT-3 attack aims to find A and A' ($\neq A$) such that the internal state collides after absorbing each associated data. Inheriting the 7-round differential trail from zero to zero difference shown in [HII+22] for the existential forgery attack, we show a CMT-3 attack with a practical complexity. Figure 8 shows the differential trail. The trail contains 25 active S-boxes. While the differential characteristic probability is 2^{-150}, an attacker knows the secret key and can compute the internal state in our attack scenario. Therefore, we do not need to rely on the probabilistic event to satisfy differential transition for 25 active S-boxes. We can choose A and A' step-by-step so that 25 active S-boxes satisfy the required differential transition.

For every active S-box out of 25 active S-boxes, only two pairs of values satisfy differential transition, and each value is (almost) fixed before every active S-box. For example, the active S-box in the 2nd round uses the differential transition, `0xF5` $\rightarrow$ `0x85`,

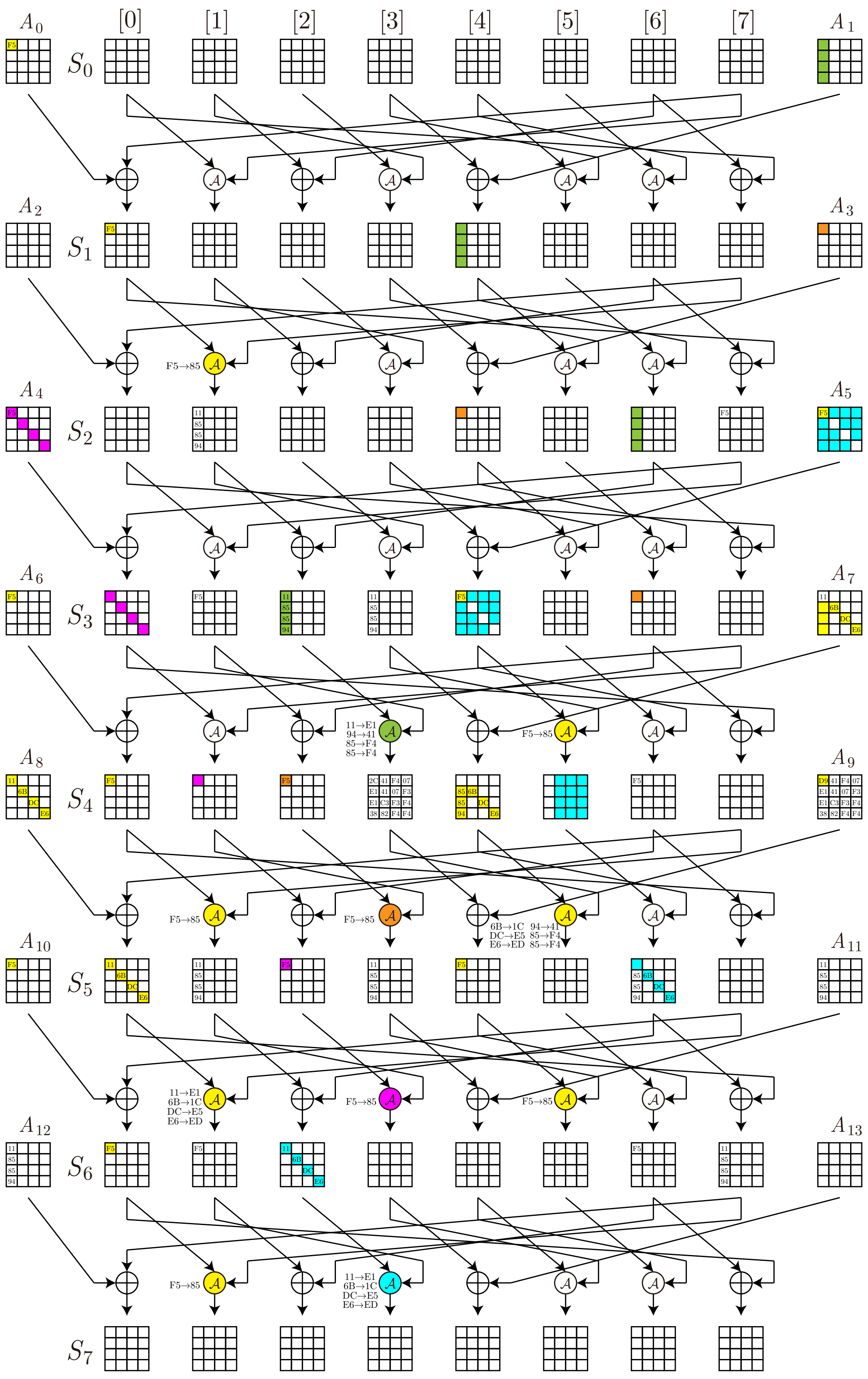

Figure 8: Differential trail for the CMT-3 attack.

Algorithm 8 Algorithm to determine A_4 and A_5 from Fig. 8

```
Input: $S_2, S_3[1..3, 5..7], S_4[0]$
Output: $A_4$, $A_5$
 1: $S_3[4]_{0,0} \leftarrow \mathrm{SboxDDT}(\mathtt{0xF5}, \mathtt{0x85})$
 2: $S_4[6]_{0,0} \leftarrow S_3[4]_{0,0} \oplus \mathcal{A}(S_3[5])_{0,0}$
 3: $S_4[1]_{0,0} \leftarrow \mathrm{SboxDDT}(\mathtt{0xF5}, \mathtt{0x85}) \oplus S_4[6]_{0,0}$
 4: $(S_3[0]_{0,0}, S_3[0]_{1,1}, S_3[0]_{2,2}) \leftarrow$ arbitrary
 5: $S_3[0]_{3,3} \leftarrow \mathrm{Sb}^{-1}[2\mathrm{Sb}[S_3[0]_{0,0}] \oplus 3\mathrm{Sb}[S_3[0]_{1,1}] \oplus \mathrm{Sb}[S_3[0]_{2,2}] \oplus S_4[1]_{0,0} \oplus S_3[7]_{0,0}]$
 6: $(\mathcal{D}_1(S_3[0]), \mathcal{D}_2(S_3[0]), \mathcal{D}_3(S_3[0])) \leftarrow$ arbitrary
 7: $A_4 \leftarrow S_3[0] \oplus S_2[7]$
 8: $(S_3[4]_{1,1}, S_3[4]_{2,2}, S_3[4]_{3,3}) \leftarrow$ arbitrary
 9: $\mathcal{C}_0(S_4[5]) \leftarrow \mathrm{MC} \circ \mathrm{SB}(\mathcal{D}_0(S_3[4])) \oplus \mathcal{C}_0(S_3[3])$
10: $(\mathcal{C}_1(S_4[5]), \mathcal{C}_2(S_4[5])) \leftarrow$ arbitrary
11: $\mathcal{D}_0(S_6[2])$ is determined by SboxDDT
12: $\mathcal{D}_0(S_5[6]) \leftarrow \mathcal{D}_0(S_6[2]) \oplus \mathcal{D}_0(\mathcal{A}(S_4[0]) \oplus S_3[6] \oplus S_3[0])$
13: $S_4[5]_{3,3} \leftarrow \mathrm{Sb}^{-1}[2\mathrm{Sb}[S_4[5]_{0,0}] \oplus 3\mathrm{Sb}[S_4[5]_{1,1}] \oplus \mathrm{Sb}[S_4[5]_{2,2}] \oplus S_4[4]_{0,0} \oplus S_5[6]_{0,0}]$
14: $S_4[5]_{2,3} \leftarrow \mathrm{Sb}^{-1}[3^{-1}(\mathrm{Sb}[S_4[5]_{0,1}] \oplus 2\mathrm{Sb}[S_4[5]_{1,2}] \oplus \mathrm{Sb}[S_4[5]_{3,0}] \oplus S_4[4]_{1,1} \oplus S_5[6]_{1,1})]$
15: $S_4[5]_{1,3} \leftarrow \mathrm{Sb}^{-1}[\mathrm{Sb}[S_4[5]_{0,2}] \oplus 2\mathrm{Sb}[S_4[5]_{2,0}] \oplus 3\mathrm{Sb}[S_4[5]_{3,1}] \oplus S_4[4]_{2,2} \oplus S_5[6]_{2,2}]$
16: $S_4[5]_{0,3} \leftarrow \mathrm{Sb}^{-1}[3^{-1}(\mathrm{Sb}[S_4[5]_{1,0}] \oplus \mathrm{Sb}[S_4[5]_{2,1}] \oplus 2\mathrm{Sb}[S_4[5]_{3,2}] \oplus S_4[4]_{3,3} \oplus S_5[6]_{3,3})]$
17: $A_5 \leftarrow \mathcal{A}^{-1}(S_4[5] \oplus S_3[3]) \oplus S_2[3]$
```

and only the following two pairs satisfy the transition.

$$(\mathtt{0x00}, \mathtt{0xF5}) \xrightarrow{\mathrm{Sb}} (\mathtt{0x63}, \mathtt{0xE6})$$
$$(\mathtt{0x42}, \mathtt{0xB7}) \xrightarrow{\mathrm{Sb}} (\mathtt{0x2C}, \mathtt{0xA9})$$

We must choose one of them and fix the S-box input, i.e., $S_1[0]_{0,0}$. For $S_1[0]_{0,0}$ to be the chosen value, we choose $A_{0,0,0}$.

We can control the value of $S_r[0]$ by choosing A_{2r-2}. Similarly, we can control the value of $S_r[4]$ by choosing A_{2r-1}. Therefore, 15 out of 25 active S-boxes (colored in yellow in Fig. 8) are controllable by choosing $A_0, A_6, A_8, A_{10}, A_5, A_7, A_9$. We still have ten active S-boxes, and these inputs are $S_r[2]$. We control $S_3[2]$ and $S_4[2]$ by choosing A_1 and A_3, respectively. We also control $S_5[2]$ by choosing A_4. Note that A_1, A_3, and A_4 are not used to control $S_r[0]$ and $S_r[4]$ in above. To control $S_6[2]$, we choose A_5. Although it is already used to control $S_3[4]$, only one active S-box needs to be controlled there, i.e., we only choose $A_{5,0,0}$ to control $S_3[4]$. On the other hand, to control the diagonal of $S_6[2]$, A_5 except for $A_{5,0,0}$ is enough.

We can find A and A' satisfying this differential trail with a complexity of $O(1)$. Specifically, we first determine A_0, A_1, A_2, and A_3 such that their related active S-boxes satisfy the differential transition. Then, we can determine S_1, S_2, and S_3 except for $S_3[0]$ and $S_3[4]$. We can also determine A_6 and A_7 such that their related active S-boxes satisfy the differential transitions. Then, we can determine $S_4[0]$ and $S_4[4]$. To find A_4 and A_5, we need to decompose each related block into bytes. Algorithm 8 shows the procedure, where $\mathcal{C}_i$ and $\mathcal{D}_i$ denote the ith column and ith diagonal of the input matrix, respectively. It is not difficult to find A_8, A_9, A_{10}, A_{11}, A_{12}, and A_{13}. We focus on related active S-box and choose each value such that related active S-boxes satisfy the differential transitions.

The attack complexity for the CMT-3 attack is $O(1)$. We implemented the attack algorithm and the test case is shown in Appendix A.

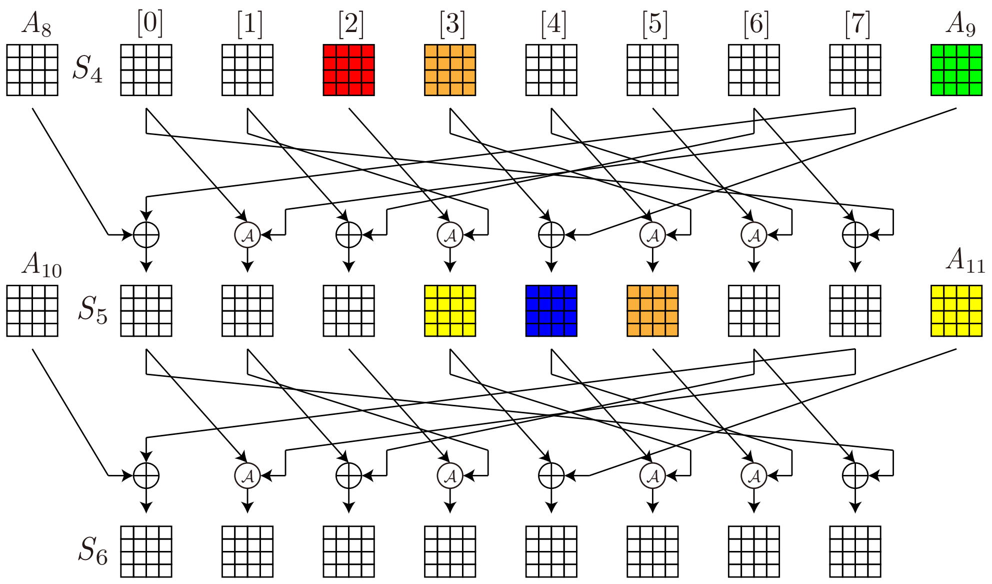

Figure 9: CMT-2, CMT-1, and FROB attacks.

5.2 CMT-2, CMT-1, and FROB Attacks

When either or both the key and nonce differ, the initialization outputs different initial states. It is unlikely that each initial state is controllable even if the key is known, considering the initialization has 20 rounds. Therefore, we show an algorithm to find A and A', where the internal state collides after absorbing each associated data from two randomly chosen initial states. Once the internal state collides before processing the message, it is trivial to demonstrate the CMT-2 attack. Note that when the finalization is independent of the key like Rocca-v1 or Rocca-v4, it can be the CMT-1 and FROB attacks.

Our attack is presented in Algorithm 9. We prepare two associated data such that 6 out of 8 blocks collide after absorbing each associated data. Specifically, $S[0,1,4..7]$ collide. We have several methods to find such associated data. As shown in Sect. 3.2, it is possible to fix the value of $S_4[0,3..7]$. When the same associated data is absorbed in an additional one round, $S_5[0,1,4..7]$ collide. It is also possible to directly fix $S_4[0,1,4..7]$. Hereinafter, we show our analysis using the state, where $S_4[0,1,4..7]$ collide.

Figure 9 shows the overview of the CMT-2, CMT-1, and FROB attacks. Now, $S_4[2]$ and $S_4[3]$ have differences, and the other 6 blocks do not have differences. After one round, $S_5[3]$, $S_5[4]$, and $S_5[5]$ have differences. It is easy to cancel out the difference in $S_6[4]$ by using A_{11} such that $\Delta A_{11} = \Delta S_5[3]$. We aim to cancel out the difference in $S_6[5]$ and $S_6[6]$ at the same time by choosing A_9.

The condition to succeed in the CMT-2 attack is

$$S_6[5] = \mathcal{A}(S_5[4]) \oplus S_5[3] = \mathcal{A}(S'_5[4]) \oplus S'_5[3]\,,$$
$$S_6[6] = \mathcal{A}(S_5[6]) \oplus S_5[4] = \mathcal{A}(S'_5[6]) \oplus S'_5[4]\,.$$

From the equations above, we have

$$\mathcal{A}^{-1}(S_6[5] \oplus S_5[3]) \oplus \mathcal{A}^{-1}(S_6[5] \oplus S'_5[3]) = S_5[4] \oplus S'_5[4] = \mathcal{A}(S_5[6]) \oplus \mathcal{A}(S'_5[6])\,.$$

We cannot choose $S_5[6]$ and $S'_5[6]$. Therefore, $S_5[4] \oplus S'_5[4]$ is determined. For any $S_6[5]$,

$$\mathrm{SB}(S_5[4]) \oplus \mathrm{SB}(S'_5[4]) = \mathrm{SR}^{-1} \circ \mathrm{MC}^{-1}(S_5[3] \oplus S'_5[3])\,,$$

Algorithm 9 Collision from two different states

```
Input: $S_0, S'_0$
Output: $A_0, \ldots, A_{11}, A'_0, \ldots, A'_{11}$
 1: Choose arbitrary $A_8$, $A_{10}$, and $A_{11}$.
 2: Set $A'_8 \leftarrow A_8$ and $A'_{10} \leftarrow A_{10}$.
 3: while $S_6 \neq S'_6$ do
 4:     Choose $(A_0, A_1), (A'_0, A'_1)$ randomly
 5:     $S_1 \leftarrow R(S_0, A_0, A_1)$
 6:     $S'_1 \leftarrow R(S'_0, A'_0, A'_1)$
 7:     Obtain $A_2, \ldots, A_7$ and $A'_2, \ldots, A'_7$ satisfying $S_4[0,1,4..7] = S'_4[0,1,4..7]$
 8:     Compute $S_5[3,5]$ and $S'_5[3,5]$
 9:     $\Delta I = \mathcal{A}(S_5[5]) \oplus \mathcal{A}(S'_5[5])$
10:     $\Delta O = \mathrm{SR}^{-1} \circ \mathrm{MC}^{-1}(S_5[3] \oplus S'_5[3])$
11:     for $(i,j) \in \{0,1,2,3\} \times \{0,1,2,3\}$ do
12:         if $\Delta I_{i,j} \xrightarrow{\mathrm{Sb}} \Delta O_{i,j}$ is possible then
13:             Pick an input $x$ s.t. $\mathrm{Sb}(x) \oplus \mathrm{Sb}(x \oplus \Delta I_{i,j}) = \Delta O_{i,j}$
14:             $A_{9,i,j} = S_4[3]_{i,j} \oplus x$
15:             $A'_{9,i,j} = S'_4[3]_{i,j} \oplus x \oplus \Delta I_{i,j}$
16:         end if
17:     end for
18:     $A'_{11} = A_{11} \oplus S_5[3] \oplus S'_5[3]$
19:     Compute $S_6$ and $S'_6$
20: end while
```

and $S_5[3] \oplus S'_5[3]$ is determined. In summary, the input and output difference of the S-box is determined, but we can freely choose $S_5[4]$ and $S'_5[4]$ by controlling A_9 and A'_9. Therefore, when the differential transition from $\Delta I = S_5[4] \oplus S'_5[4] = \mathcal{A}(S_5[5]) \oplus \mathcal{A}(S'_5[5])$ to $\Delta O = \mathrm{SR}^{-1} \circ \mathrm{MC}^{-1}(S_5[3] \oplus S'_5[3])$ is possible, we can choose such $S_5[4]$ and $S'_5[4]$.

The input and output differences of the S-box are determined once we construct a pair such that $S_4[0,1,4..7]$ collide. The probability that randomly chosen input/output differences are possible is about 1/2. Since there are 16 S-boxes, the probability that we can construct such A_9 and A'_9 is 2^{-16}. In our CMT-2 attack, we construct such (S_4, S'_4), and if it does not lead to a possible differential transition, we reconstruct different (S_4, S'_4) until we have a pair having a possible transition. Therefore, the attack complexity is 2^{16}. We emphasize that the complexity is practical.

CMT-1 and FROB Attacks. The CMT-2 attack above can be applied regardless of the keyed/non-keyed finalization. When the finalization does not involve the key, the internal state collision leads to the same ciphertext and tag, even for the CMT-1 and FROB attacks. Therefore, our CMT-2 attack is converted to the CMT-1 and FROB attacks against Rocca with non-keyed finalization like Rocca-v1 or Rocca-v4. To the best of our knowledge, there is no existing FROB attack against Rocca that is faster than the generic complexity, 2^{64}. In a recently published attack in ToSC 2024(1)/FSE 2024 [DFI+24], the attack complexity is 2^{128}, which is worse than the generic attack. We emphasize that our FROB attack is practical and significantly improves the existing attack. We implemented our algorithm and the test case of the FROB attack against Rocca-v4 is shown in Appendix B.

6 Conclusions

In this paper, we analysed revised versions of Rocca with various finalization, and clarified security trade-offs. For committing attacks, none of the versions of Rocca is secure, while

Rocca-v1 and Rocca-v4 allow a practical attack in FROB notion, one of the strongest notions of committing attacks. Our result significantly improves the result in [DFI+24] on Rocca-v1. These schemes cannot be used in applications where committing security is expected. XORing the key at the beginning of finalization in Rocca-v2 and Rocca-v3 is supposed to add resistance against universal forgery, while this is the crucial reason why these schemes allow key recovery that is faster than the exhaustive key search. With respect to universal forgery in the nonce-misuse setting, Rocca-v1 is worse than GCM in that it allows key recovery, and Rocca-v4 inherits the weakness of GCM allowing practical universal forgery. For Rocca-v2 and Rocca-v3, they admit universal forgery with a practical data complexity, while this not a concern for the high time complexity.

In the security notions in the secret key setting, i.e., key recovery and universal forgery, whether a nonce is reused or respected, Rocca-v2 and Rocca-v3 do not reach the level of their expected security, while none of the attacks we presented is a practical concern. The design of Rocca-v4 can be interpreted to introduce a risk of practical universal forgery in the nonce-misuse setting to avoid the impractical attacks on Rocca-v2 and Rocca-v3.

Rocca-S [ABC+23] is another revised version of Rocca to address the attacks presented in [HII+22]. The design follows Rocca-v4 in that XORing the key in the finalization is omitted, and it would be interesting to study its security in light of our results of this paper.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments and suggestions, especially for pointing out the generic universal forgery discussed in Sect. 4.3.1. We also would like to thank the designers of Rocca for their feedback on the preliminary draft of the paper, and the authors of [DFI+24] for comments. This work was supported in part by JSPS KAKENHI Grant Number JP24K07489.

References

[ABC+23] Ravi Anand, Subhadeep Banik, Andrea Caforio, Kazuhide Fukushima, Takanori Isobe, Shinsaku Kiyomoto, Fukang Liu, Yuto Nakano, Kosei Sakamoto, and Nobuyuki Takeuchi. An ultra-high throughput AES-based authenticated encryption scheme for 6G: Design and implementation. In Gene Tsudik, Mauro Conti, Kaitai Liang, and Georgios Smaragdakis, editors, *Computer Security - ESORICS 2023 - 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25-29, 2023, Proceedings, Part I*, volume 14344 of *Lecture Notes in Computer Science*, pages 229–248. Springer, 2023.

[ADG+22] Ange Albertini, Thai Duong, Shay Gueron, Stefan Kölbl, Atul Luykx, and Sophie Schmieg. How to abuse and fix authenticated encryption without key commitment. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 3291–3308. USENIX Association, 2022.

[ADL17] Tomer Ashur, Orr Dunkelman, and Atul Luykx. Boosting authenticated encryption robustness with minimal modifications. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2017.

[AES01] Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce, November 2001.

[AI21] Ravi Anand and Takanori Isobe. Differential fault attack on Rocca. In Jong Hwan Park and Seung-Hyun Seo, editors, *Information Security and Cryptology - ICISC 2021 - 24th International Conference, Seoul, South Korea, December 1-3, 2021, Revised Selected Papers*, volume 13218 of *Lecture Notes in Computer Science*, pages 283–295. Springer, 2021.

[AI23] Ravi Anand and Takanori Isobe. Quantum security analysis of Rocca. *Quantum Inf. Process.*, 22(4):164, 2023.

[BH22] Mihir Bellare and Viet Tung Hoang. Efficient schemes for committing authenticated encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 845–875. Springer, 2022.

[BS23] Xavier Bonnetain and André Schrottenloher. Single-query quantum hidden shift attacks. *IACR Cryptol. ePrint Arch., 2023/1306*, 2023.

[BZD$^+$16] Hanno Böck, Aaron Zauner, Sean Devlin, Juraj Somorovsky, and Philipp Jovanovic. Nonce-disrespecting adversaries: Practical forgery attacks on GCM in TLS. In Natalie Silvanovich and Patrick Traynor, editors, *10th USENIX Workshop on Offensive Technologies, WOOT 16, Austin, TX, USA, August 8-9, 2016.* USENIX Association, 2016.

[CAE] CAESAR. Competition for authenticated encryption: Security, applicability, and robustness.

[CR22] John Chan and Phillip Rogaway. On committing authenticated-encryption. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part II*, volume 13555 of *Lecture Notes in Computer Science*, pages 275–294. Springer, 2022.

[DFI$^+$24] Patrick Derbez, Pierre-Alain Fouque, Takanori Isobe, Mostafizar Rahman, and André Schrottenloher. Key committing attacks against AES-based AEAD schemes. *IACR Trans. Symmetric Cryptol.*, 2024(1):135–157, 2024.

[DGRW18] Yevgeniy Dodis, Paul Grubbs, Thomas Ristenpart, and Joanne Woodage. Fast message franking: From invisible salamanders to encryptment. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 155–186. Springer, 2018.

[FOR17] Pooya Farshim, Claudio Orlandi, and Razvan Rosie. Security of symmetric primitives under incorrect usage of keys. *IACR Trans. Symmetric Cryptol.*, 2017(1):449–473, 2017.

[GL15] Shay Gueron and Yehuda Lindell. GCM-SIV: full nonce misuse-resistant authenticated encryption at under one cycle per byte. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC*

Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015, pages 109–119. ACM, 2015.

[GLR17] Paul Grubbs, Jiahui Lu, and Thomas Ristenpart. Message franking via committing authenticated encryption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 66–97. Springer, 2017.

[Gue10] Shay Gueron. Intel Advanced Encryption Standard (AES) New Instructions Set. 2010.

[HII$^+$22] Akinori Hosoyamada, Akiko Inoue, Ryoma Ito, Tetsu Iwata, Kazuhiko Mimematsu, Ferdinand Sibleyras, and Yosuke Todo. Cryptanalysis of Rocca and feasibility of its security claim. *IACR Trans. Symmetric Cryptol.*, 2022(3):123–151, 2022.

[JN16] Jérémy Jean and Ivica Nikolic. Efficient design strategies based on the AES round function. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 334–353. Springer, 2016.

[Jou06] Antoine Joux. Authentication failures in NIST version of GCM. Public comment to NIST, 2006.

[LGR21] Julia Len, Paul Grubbs, and Thomas Ristenpart. Partitioning oracle attacks. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, pages 195–212. USENIX Association, 2021.

[MLGR23] Sanketh Menda, Julia Len, Paul Grubbs, and Thomas Ristenpart. Context discovery and commitment attacks - how to break CCM, EAX, SIV, and more. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part IV*, volume 14007 of *Lecture Notes in Computer Science*, pages 379–407. Springer, 2023.

[MV04] David A. McGrew and John Viega. The security and performance of the Galois/Counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *Progress in Cryptology - INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20-22, 2004, Proceedings*, volume 3348 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2004.

[Nik14] Ivica Nikolic. Tiaoxin-346: Version 2.0. CAESAR Competition, 2014.

[PS16] Thomas Peyrin and Yannick Seurin. Counter-in-Tweak: Authenticated encryption modes for tweakable block ciphers. 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 I*, volume 9814 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2016.

[RKHP20] David Rupprecht, Katharina Kohls, Thorsten Holz, and Christina Pöpper. Call me maybe: Eavesdropping encrypted LTE calls with ReVoLTE. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 73–88. USENIX Association, 2020.

[SLN+21] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient AES-based encryption scheme for beyond 5G. *IACR Trans. Symmetric Cryptol.*, 2021(2):1–30, 2021.

[SLN+22a] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient AES-based encryption scheme for beyond 5G (full version). *IACR Cryptol. ePrint Arch., 2022/116*, 2022. Version 20220914.

[SLN+22b] Kosei Sakamoto, Fukang Liu, Yuto Nakano, Shinsaku Kiyomoto, and Takanori Isobe. Rocca: An efficient AES-based encryption scheme for beyond 5G (full version). *IACR Cryptol. ePrint Arch., 2022/116*, 2022. Version 20230316.

[STSI23] Takuro Shiraya, Nobuyuki Takeuchi, Kosei Sakamoto, and Takanori Isobe. MILP-based security evaluation for AEGIS/Tiaoxin-346/Rocca. *IET Inf. Secur.*, 17(3):458–467, 2023.

[TI23] Ryunosuke Takeuchi and Tetsu Iwata. Key recovery attack against modified version of Rocca. Private Communication, 2023.

[TSI23a] Nobuyuki Takeuchi, Kosei Sakamoto, and Takanori Isobe. On optimality of the round function of Rocca. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 106(1):45–53, 2023.

[TSI23b] Nobuyuki Takeuchi, Kosei Sakamoto, and Takanori Isobe. Security evaluation of initialization phases and round functions of Rocca and AEGIS. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 106(3):253–262, 2023.

[VP17] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1313–1328. ACM, 2017.

[WP13] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 185–201. Springer, 2013.

A Test Case for the CMT-3 Attack

We present a test case for the committing attack. We consider Rocca-v3 [SLN+22a]. We present a concrete example of (K, N, A, M), (K', N', A', M'), and (C, T) such that $(C, T) = \mathrm{Enc}_K(N, A, M) = \mathrm{Enc}_{K'}(N', A', M')$, with the constraint that $(K, N, A) \neq (K', N', A')$.

We define K, K', N, and N' as follows (written in hex in an array):

$$\begin{aligned}
K_0 = K'_0 &= \{\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01}\}\\
K_1 = K'_1 &= \{\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01},\texttt{01}\}\\
N = N' &= \{\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02},\texttt{02}\}
\end{aligned}$$

The state after the initialization of Rocca-v3 becomes S_0 as follows:

$$\begin{aligned}
S_0[0] &= \{\mathtt{7C, D6, 06, 78, 19, 67, FA, D3, F0, 85, 36, 0D, 45, 6A, 48, E7}\}\\
S_0[1] &= \{\mathtt{1E, 17, 6A, 84, 76, 94, 30, B0, 99, 33, A3, 79, 11, AC, A3, 0C}\}\\
S_0[2] &= \{\mathtt{49, 57, E3, DD, CE, 65, 99, D0, D0, C5, C0, E2, 0A, E2, 17, FB}\}\\
S_0[3] &= \{\mathtt{4E, D8, 09, 31, 7D, 12, 56, CC, 11, 48, F8, EB, F8, 50, 56, CD}\}\\
S_0[4] &= \{\mathtt{80, A1, C7, 5D, CC, 33, 55, 39, B0, 16, D9, 1C, 72, F2, 0A, 72}\}\\
S_0[5] &= \{\mathtt{43, 9A, AE, 0C, 20, B9, 8D, 3A, 54, ED, 7E, BA, FC, 20, 9C, 88}\}\\
S_0[6] &= \{\mathtt{8D, 70, CF, 5B, 7E, CA, BB, 60, 35, 76, 18, 8D, 61, 3C, 82, E2}\}\\
S_0[7] &= \{\mathtt{14, E8, 6D, 10, B3, 2A, 4E, B3, 86, 99, D8, B6, 09, 56, DF, 5B}\}
\end{aligned}$$

Then, the associated data A and A' obtained by the method shown in Sect. 5.1 are as follows:

$$\begin{aligned}
A_0 &= \{\mathtt{14, 89, C0, 47, EB, CB, 90, ED, 8C, 85, 81, FC, 48, 4F, CC, 86}\}\\
A_1 &= \{\mathtt{AC, 0A, D3, C6, B2, 92, EA, 32, 14, D4, 9C, 44, FE, 8B, 9D, D7}\}\\
A_2 &= \{\mathtt{D7, DE, E9, 3D, E9, 60, 89, 16, F1, 56, 07, 39, F7, 8D, 3F, 01}\}\\
A_3 &= \{\mathtt{63, D8, EA, 47, 3C, 8D, 79, 83, D1, 73, C8, C6, 08, AF, 1E, 95}\}\\
A_4 &= \{\mathtt{31, 18, 1A, 73, 39, F1, 34, 29, 03, 2B, 50, D2, C4, 4C, 6D, ED}\}\\
A_5 &= \{\mathtt{B8, 4E, F9, 2A, 39, D7, 3C, 61, 44, B1, 16, D2, 92, 2E, 61, 17}\}\\
A_6 &= \{\mathtt{08, 39, 20, 8F, 64, 53, A2, EF, 9B, A6, 0A, 57, A5, 88, 17, 19}\}\\
A_7 &= \{\mathtt{C7, FD, D1, B0, F3, B8, B7, 03, 5C, A0, 71, 80, 4A, D8, 71, 98}\}\\
A_8 &= \{\mathtt{4F, D5, DB, 4D, CD, 63, 95, C8, 06, 02, 59, 4B, 34, 4C, 07, 9B}\}\\
A_9 &= \{\mathtt{31, 67, 39, 60, 14, 07, F0, 0D, 1E, 2D, FC, 0A, 24, C4, D2, 17}\}\\
A_{10} &= \{\mathtt{FB, 98, A0, 3C, 51, CA, 8C, E9, 87, DC, C1, 98, 79, 94, 5E, 52}\}\\
A_{11} &= \{\mathtt{73, DC, B6, D6, 61, 67, 60, C6, 82, 62, E9, C2, EC, 0A, 52, 47}\}\\
A_{12} &= \{\mathtt{E7, 85, 76, 8D, B0, 91, B0, F1, 36, AA, 16, 72, 0F, 14, E5, 61}\}\\
A_{13} &= \{\mathtt{B5, B7, 95, D3, 54, 3D, E7, 19, 0A, 38, 60, F1, 60, D9, F9, 8E}\}
\end{aligned}$$

$$\begin{aligned}
A'_0 &= \{\mathtt{E1, 89, C0, 47, EB, CB, 90, ED, 8C, 85, 81, FC, 48, 4F, CC, 86}\}\\
A'_1 &= \{\mathtt{AC, 0A, D3, C6, B2, 92, EA, 32, 14, D4, 9C, 44, FE, 8B, 9D, D7}\}\\
A'_2 &= \{\mathtt{D7, DE, E9, 3D, E9, 60, 89, 16, F1, 56, 07, 39, F7, 8D, 3F, 01}\}\\
A'_3 &= \{\mathtt{63, D8, EA, 47, 3C, 8D, 79, 83, D1, 73, C8, C6, 08, AF, 1E, 95}\}\\
A'_4 &= \{\mathtt{C4, 18, 1A, 73, 39, F1, 34, 29, 03, 2B, 50, D2, C4, 4C, 6D, ED}\}\\
A'_5 &= \{\mathtt{4D, 4E, F9, 2A, 39, D7, 3C, 61, 44, B1, 16, D2, 92, 2E, 61, 17}\}\\
A'_6 &= \{\mathtt{FD, 39, 20, 8F, 64, 53, A2, EF, 9B, A6, 0A, 57, A5, 88, 17, 19}\}\\
A'_7 &= \{\mathtt{D6, FD, D1, B0, F3, D3, B7, 03, 5C, A0, AD, 80, 4A, D8, 71, 7E}\}\\
A'_8 &= \{\mathtt{5E, D5, DB, 4D, CD, 08, 95, C8, 06, 02, 85, 4B, 34, 4C, 07, 7D}\}\\
A'_9 &= \{\mathtt{E8, 86, D8, 58, 55, 46, 33, 8F, EA, 2A, 0F, FE, 23, 37, 26, E3}\}\\
A'_{10} &= \{\mathtt{0E, 98, A0, 3C, 51, CA, 8C, E9, 87, DC, C1, 98, 79, 94, 5E, 52}\}\\
A'_{11} &= \{\mathtt{62, 59, 33, 42, 61, 67, 60, C6, 82, 62, E9, C2, EC, 0A, 52, 47}\}\\
A'_{12} &= \{\mathtt{F6, 00, F3, 19, B0, 91, B0, F1, 36, AA, 16, 72, 0F, 14, E5, 61}\}\\
A'_{13} &= \{\mathtt{B5, B7, 95, D3, 54, 3D, E7, 19, 0A, 38, 60, F1, 60, D9, F9, 8E}\}
\end{aligned}$$

We also let the messages $M = (M_0, M_1)$ and $M' = (M'_0, M'_1)$ be the following values:

$$M_0 = M'_0 = \{\texttt{FF}, \texttt{FF}, \texttt{EE}, \texttt{EE}, \texttt{DD}, \texttt{DD}, \texttt{CC}, \texttt{CC}, \texttt{BB}, \texttt{BB}, \texttt{AA}, \texttt{AA}, \texttt{99}, \texttt{99}, \texttt{88}, \texttt{88}\}$$
$$M_1 = M'_1 = \{\texttt{77}, \texttt{77}, \texttt{66}, \texttt{66}, \texttt{55}, \texttt{55}, \texttt{44}, \texttt{44}, \texttt{33}, \texttt{33}, \texttt{22}, \texttt{22}, \texttt{11}, \texttt{11}, \texttt{00}, \texttt{00}\}$$

Then, the outputs of $\mathrm{Enc}_K(N, A, M)$ and $\mathrm{Enc}_{K'}(N', A', M')$ are identical, i.e., we have $(C, T) = \mathrm{Enc}_K(N, A, M) = \mathrm{Enc}_{K'}(N', A', M')$, where $C = (C_0, C_1)$ and

$$C_0 = \{\texttt{9B}, \texttt{2E}, \texttt{73}, \texttt{39}, \texttt{8E}, \texttt{37}, \texttt{D2}, \texttt{80}, \texttt{50}, \texttt{EF}, \texttt{C1}, \texttt{72}, \texttt{1F}, \texttt{7F}, \texttt{E7}, \texttt{99}\},$$
$$C_1 = \{\texttt{8B}, \texttt{6D}, \texttt{C1}, \texttt{2C}, \texttt{85}, \texttt{D6}, \texttt{9D}, \texttt{70}, \texttt{2F}, \texttt{15}, \texttt{EB}, \texttt{C2}, \texttt{46}, \texttt{82}, \texttt{2E}, \texttt{DA}\},$$
$$T = \{\texttt{1B}, \texttt{0C}, \texttt{98}, \texttt{98}, \texttt{8B}, \texttt{01}, \texttt{AC}, \texttt{5C}, \texttt{88}, \texttt{94}, \texttt{EF}, \texttt{77}, \texttt{95}, \texttt{D3}, \texttt{81}, \texttt{CB}\}.$$

As a result, we obtain $(K, N, A, M), (K', N', A', M')$, and (C, T) such that $(C, T) = \mathrm{Enc}_K(N, A, M) = \mathrm{Enc}_{K'}(N', A', M')$, $A \neq A'$, and $(K, N, M) = (K', N', M')$, practically breaking the CMT-3 security of Rocca-v3 in [SLN+22a].

B Test Case for the FROB Attack

We present a test case for the FROB attack. We consider Rocca-v4 in [SLN+22b]. We present an example of $(K, N, A, M), (K', N', A', M')$, and (C, T) such that $(C, T) = \mathrm{Enc}_K(N, A, M) = \mathrm{Enc}_{K'}(N', A', M')$, with the constraint that $K \neq K', N = N'$.

We define K, K', N, and N' as follows (written in hex in an array):

$$K_0 = \{\texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}\}$$
$$K_1 = \{\texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}, \texttt{01}\}$$
$$K'_0 = \{\texttt{01}, \texttt{23}, \texttt{45}, \texttt{67}, \texttt{89}, \texttt{AB}, \texttt{CD}, \texttt{EF}, \texttt{01}, \texttt{23}, \texttt{45}, \texttt{67}, \texttt{89}, \texttt{AB}, \texttt{CD}, \texttt{EF}\}$$
$$K'_1 = \{\texttt{01}, \texttt{23}, \texttt{45}, \texttt{67}, \texttt{89}, \texttt{AB}, \texttt{CD}, \texttt{EF}, \texttt{01}, \texttt{23}, \texttt{45}, \texttt{67}, \texttt{89}, \texttt{AB}, \texttt{CD}, \texttt{EF}\}$$
$$N = N' = \{\texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}, \texttt{02}\}$$

The state after the initialization of Rocca-v4 becomes S_0, S'_0 as follows:

$$S_0[0] = \{\texttt{7C}, \texttt{D6}, \texttt{06}, \texttt{78}, \texttt{19}, \texttt{67}, \texttt{FA}, \texttt{D3}, \texttt{F0}, \texttt{85}, \texttt{36}, \texttt{0D}, \texttt{45}, \texttt{6A}, \texttt{48}, \texttt{E7}\}$$
$$S_0[1] = \{\texttt{1E}, \texttt{17}, \texttt{6A}, \texttt{84}, \texttt{76}, \texttt{94}, \texttt{30}, \texttt{B0}, \texttt{99}, \texttt{33}, \texttt{A3}, \texttt{79}, \texttt{11}, \texttt{AC}, \texttt{A3}, \texttt{0C}\}$$
$$S_0[2] = \{\texttt{49}, \texttt{57}, \texttt{E3}, \texttt{DD}, \texttt{CE}, \texttt{65}, \texttt{99}, \texttt{D0}, \texttt{D0}, \texttt{C5}, \texttt{C0}, \texttt{E2}, \texttt{0A}, \texttt{E2}, \texttt{17}, \texttt{FB}\}$$
$$S_0[3] = \{\texttt{4E}, \texttt{D8}, \texttt{09}, \texttt{31}, \texttt{7D}, \texttt{12}, \texttt{56}, \texttt{CC}, \texttt{11}, \texttt{48}, \texttt{F8}, \texttt{EB}, \texttt{F8}, \texttt{50}, \texttt{56}, \texttt{CD}\}$$
$$S_0[4] = \{\texttt{80}, \texttt{A1}, \texttt{C7}, \texttt{5D}, \texttt{CC}, \texttt{33}, \texttt{55}, \texttt{39}, \texttt{B0}, \texttt{16}, \texttt{D9}, \texttt{1C}, \texttt{72}, \texttt{F2}, \texttt{0A}, \texttt{72}\}$$
$$S_0[5] = \{\texttt{43}, \texttt{9A}, \texttt{AE}, \texttt{0C}, \texttt{20}, \texttt{B9}, \texttt{8D}, \texttt{3A}, \texttt{54}, \texttt{ED}, \texttt{7E}, \texttt{BA}, \texttt{FC}, \texttt{20}, \texttt{9C}, \texttt{88}\}$$
$$S_0[6] = \{\texttt{8D}, \texttt{70}, \texttt{CF}, \texttt{5B}, \texttt{7E}, \texttt{CA}, \texttt{BB}, \texttt{60}, \texttt{35}, \texttt{76}, \texttt{18}, \texttt{8D}, \texttt{61}, \texttt{3C}, \texttt{82}, \texttt{E2}\}$$
$$S_0[7] = \{\texttt{14}, \texttt{E8}, \texttt{6D}, \texttt{10}, \texttt{B3}, \texttt{2A}, \texttt{4E}, \texttt{B3}, \texttt{86}, \texttt{99}, \texttt{D8}, \texttt{B6}, \texttt{09}, \texttt{56}, \texttt{DF}, \texttt{5B}\}$$

$$S'_0[0] = \{\texttt{DB}, \texttt{BB}, \texttt{B6}, \texttt{E5}, \texttt{D3}, \texttt{5C}, \texttt{25}, \texttt{1F}, \texttt{4D}, \texttt{05}, \texttt{4A}, \texttt{F3}, \texttt{60}, \texttt{EF}, \texttt{25}, \texttt{3D}\}$$
$$S'_0[1] = \{\texttt{D7}, \texttt{D4}, \texttt{09}, \texttt{EB}, \texttt{4E}, \texttt{78}, \texttt{98}, \texttt{E9}, \texttt{0E}, \texttt{D7}, \texttt{77}, \texttt{03}, \texttt{92}, \texttt{19}, \texttt{45}, \texttt{DC}\}$$
$$S'_0[2] = \{\texttt{73}, \texttt{8E}, \texttt{E3}, \texttt{F2}, \texttt{F7}, \texttt{82}, \texttt{24}, \texttt{EA}, \texttt{12}, \texttt{AA}, \texttt{14}, \texttt{1B}, \texttt{2C}, \texttt{1A}, \texttt{72}, \texttt{9F}\}$$
$$S'_0[3] = \{\texttt{C5}, \texttt{A4}, \texttt{15}, \texttt{04}, \texttt{07}, \texttt{1D}, \texttt{7A}, \texttt{00}, \texttt{12}, \texttt{A1}, \texttt{1B}, \texttt{67}, \texttt{E0}, \texttt{4E}, \texttt{C1}, \texttt{82}\}$$
$$S'_0[4] = \{\texttt{7C}, \texttt{54}, \texttt{41}, \texttt{0F}, \texttt{47}, \texttt{A7}, \texttt{D2}, \texttt{19}, \texttt{70}, \texttt{67}, \texttt{0E}, \texttt{EA}, \texttt{97}, \texttt{3C}, \texttt{91}, \texttt{3A}\}$$
$$S'_0[5] = \{\texttt{44}, \texttt{63}, \texttt{A1}, \texttt{8D}, \texttt{06}, \texttt{E0}, \texttt{9D}, \texttt{C6}, \texttt{FA}, \texttt{A9}, \texttt{AE}, \texttt{49}, \texttt{DA}, \texttt{C6}, \texttt{92}, \texttt{4B}\}$$
$$S'_0[6] = \{\texttt{69}, \texttt{AF}, \texttt{6D}, \texttt{4D}, \texttt{1E}, \texttt{C6}, \texttt{F4}, \texttt{8B}, \texttt{CC}, \texttt{EF}, \texttt{FC}, \texttt{8C}, \texttt{21}, \texttt{D5}, \texttt{67}, \texttt{3F}\}$$
$$S'_0[7] = \{\texttt{00}, \texttt{70}, \texttt{81}, \texttt{33}, \texttt{35}, \texttt{C6}, \texttt{63}, \texttt{0A}, \texttt{6D}, \texttt{6D}, \texttt{0A}, \texttt{24}, \texttt{5B}, \texttt{2A}, \texttt{84}, \texttt{04}\}$$

The associated data A and A' obtained by the method shown in Sect. 5.2 are as follows:

$$\begin{aligned}
A_0 &= \{\mathtt{AC}, \mathtt{B4}, \mathtt{4A}, \mathtt{E0}, \mathtt{F6}, \mathtt{08}, \mathtt{B2}, \mathtt{FB}, \mathtt{01}, \mathtt{4D}, \mathtt{C8}, \mathtt{54}, \mathtt{3A}, \mathtt{D7}, \mathtt{CB}, \mathtt{24}\}\\
A_1 &= \{\mathtt{DC}, \mathtt{AC}, \mathtt{0D}, \mathtt{0C}, \mathtt{71}, \mathtt{70}, \mathtt{46}, \mathtt{D6}, \mathtt{C7}, \mathtt{C1}, \mathtt{8F}, \mathtt{50}, \mathtt{CB}, \mathtt{D9}, \mathtt{70}, \mathtt{5E}\}\\
A_2 &= \{\mathtt{A3}, \mathtt{10}, \mathtt{1E}, \mathtt{A0}, \mathtt{F1}, \mathtt{46}, \mathtt{A6}, \mathtt{30}, \mathtt{C4}, \mathtt{9A}, \mathtt{EA}, \mathtt{B6}, \mathtt{B2}, \mathtt{0A}, \mathtt{B3}, \mathtt{EC}\}\\
A_3 &= \{\mathtt{7D}, \mathtt{05}, \mathtt{A3}, \mathtt{98}, \mathtt{76}, \mathtt{59}, \mathtt{55}, \mathtt{74}, \mathtt{88}, \mathtt{78}, \mathtt{67}, \mathtt{56}, \mathtt{18}, \mathtt{AC}, \mathtt{79}, \mathtt{93}\}\\
A_4 &= \{\mathtt{5E}, \mathtt{03}, \mathtt{90}, \mathtt{E8}, \mathtt{41}, \mathtt{B2}, \mathtt{CC}, \mathtt{EF}, \mathtt{53}, \mathtt{DD}, \mathtt{3C}, \mathtt{61}, \mathtt{CC}, \mathtt{A4}, \mathtt{91}, \mathtt{CE}\}\\
A_5 &= \{\mathtt{C9}, \mathtt{75}, \mathtt{77}, \mathtt{EE}, \mathtt{C9}, \mathtt{00}, \mathtt{47}, \mathtt{3D}, \mathtt{D1}, \mathtt{32}, \mathtt{39}, \mathtt{FF}, \mathtt{7B}, \mathtt{2F}, \mathtt{6C}, \mathtt{14}\}\\
A_6 &= \{\mathtt{0C}, \mathtt{B9}, \mathtt{8E}, \mathtt{65}, \mathtt{7F}, \mathtt{D5}, \mathtt{31}, \mathtt{A9}, \mathtt{E9}, \mathtt{27}, \mathtt{E4}, \mathtt{6D}, \mathtt{46}, \mathtt{DD}, \mathtt{B3}, \mathtt{E4}\}\\
A_7 &= \{\mathtt{65}, \mathtt{34}, \mathtt{82}, \mathtt{ED}, \mathtt{ED}, \mathtt{4A}, \mathtt{AF}, \mathtt{EA}, \mathtt{22}, \mathtt{F0}, \mathtt{75}, \mathtt{8B}, \mathtt{1F}, \mathtt{10}, \mathtt{A9}, \mathtt{CC}\}\\
A_8 &= \{\mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}\}\\
A_9 &= \{\mathtt{DE}, \mathtt{1D}, \mathtt{6C}, \mathtt{96}, \mathtt{EE}, \mathtt{CB}, \mathtt{59}, \mathtt{B7}, \mathtt{EB}, \mathtt{58}, \mathtt{A0}, \mathtt{DE}, \mathtt{48}, \mathtt{C4}, \mathtt{A5}, \mathtt{B6}\}\\
A_{10} &= \{\mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}\}\\
A_{11} &= \{\mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}\}
\end{aligned}$$

$$\begin{aligned}
A'_0 &= \{\mathtt{BA}, \mathtt{81}, \mathtt{0A}, \mathtt{CE}, \mathtt{50}, \mathtt{BE}, \mathtt{55}, \mathtt{3D}, \mathtt{2E}, \mathtt{71}, \mathtt{C1}, \mathtt{9C}, \mathtt{58}, \mathtt{00}, \mathtt{43}, \mathtt{C9}\}\\
A'_1 &= \{\mathtt{D7}, \mathtt{0A}, \mathtt{1E}, \mathtt{B5}, \mathtt{64}, \mathtt{F0}, \mathtt{B8}, \mathtt{EB}, \mathtt{55}, \mathtt{9B}, \mathtt{0B}, \mathtt{02}, \mathtt{61}, \mathtt{D6}, \mathtt{89}, \mathtt{8B}\}\\
A'_2 &= \{\mathtt{AE}, \mathtt{A8}, \mathtt{40}, \mathtt{B0}, \mathtt{5C}, \mathtt{66}, \mathtt{35}, \mathtt{C1}, \mathtt{CB}, \mathtt{05}, \mathtt{E3}, \mathtt{8C}, \mathtt{3C}, \mathtt{11}, \mathtt{DD}, \mathtt{E6}\}\\
A'_3 &= \{\mathtt{87}, \mathtt{35}, \mathtt{B7}, \mathtt{D1}, \mathtt{2D}, \mathtt{D2}, \mathtt{C9}, \mathtt{B9}, \mathtt{16}, \mathtt{AB}, \mathtt{43}, \mathtt{52}, \mathtt{28}, \mathtt{DC}, \mathtt{4C}, \mathtt{81}\}\\
A'_4 &= \{\mathtt{51}, \mathtt{58}, \mathtt{60}, \mathtt{C3}, \mathtt{37}, \mathtt{D2}, \mathtt{8E}, \mathtt{A8}, \mathtt{51}, \mathtt{21}, \mathtt{2A}, \mathtt{FD}, \mathtt{BE}, \mathtt{F4}, \mathtt{31}, \mathtt{B0}\}\\
A'_5 &= \{\mathtt{DA}, \mathtt{BA}, \mathtt{E4}, \mathtt{49}, \mathtt{BB}, \mathtt{54}, \mathtt{A2}, \mathtt{47}, \mathtt{C2}, \mathtt{DB}, \mathtt{38}, \mathtt{65}, \mathtt{9B}, \mathtt{65}, \mathtt{B1}, \mathtt{BD}\}\\
A'_6 &= \{\mathtt{1E}, \mathtt{73}, \mathtt{24}, \mathtt{10}, \mathtt{03}, \mathtt{76}, \mathtt{F9}, \mathtt{68}, \mathtt{02}, \mathtt{12}, \mathtt{FA}, \mathtt{31}, \mathtt{54}, \mathtt{F4}, \mathtt{03}, \mathtt{80}\}\\
A'_7 &= \{\mathtt{9B}, \mathtt{09}, \mathtt{1F}, \mathtt{F8}, \mathtt{42}, \mathtt{EF}, \mathtt{97}, \mathtt{83}, \mathtt{0F}, \mathtt{D3}, \mathtt{00}, \mathtt{C5}, \mathtt{98}, \mathtt{BB}, \mathtt{73}, \mathtt{EA}\}\\
A'_8 &= \{\mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}\}\\
A'_9 &= \{\mathtt{30}, \mathtt{EA}, \mathtt{F2}, \mathtt{A0}, \mathtt{37}, \mathtt{D3}, \mathtt{0C}, \mathtt{08}, \mathtt{88}, \mathtt{52}, \mathtt{13}, \mathtt{DA}, \mathtt{B6}, \mathtt{CA}, \mathtt{D1}, \mathtt{7F}\}\\
A'_{10} &= \{\mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}, \mathtt{00}\}\\
A'_{11} &= \{\mathtt{1A}, \mathtt{D0}, \mathtt{C3}, \mathtt{1C}, \mathtt{E7}, \mathtt{F1}, \mathtt{8C}, \mathtt{EC}, \mathtt{0C}, \mathtt{AE}, \mathtt{E8}, \mathtt{4B}, \mathtt{F5}, \mathtt{39}, \mathtt{3B}, \mathtt{8C}\}
\end{aligned}$$

Let the messages $M = (M_0, M_1)$ and $M' = (M'_0, M'_1)$ be the following values:

$$\begin{aligned}
M_0 = M'_0 &= \{\mathtt{FE}, \mathtt{DC}, \mathtt{BA}, \mathtt{98}, \mathtt{76}, \mathtt{54}, \mathtt{32}, \mathtt{10}, \mathtt{FE}, \mathtt{DC}, \mathtt{BA}, \mathtt{98}, \mathtt{76}, \mathtt{54}, \mathtt{32}, \mathtt{10}\}\\
M_1 = M'_1 &= \{\mathtt{FE}, \mathtt{DC}, \mathtt{BA}, \mathtt{98}, \mathtt{76}, \mathtt{54}, \mathtt{32}, \mathtt{10}, \mathtt{FE}, \mathtt{DC}, \mathtt{BA}, \mathtt{98}, \mathtt{76}, \mathtt{54}, \mathtt{32}, \mathtt{10}\}
\end{aligned}$$

Then, the outputs of $\mathrm{Enc}_K(N, A, M)$ and $\mathrm{Enc}_{K'}(N', A', M')$ are identical, i.e., we have $(C, T) = \mathrm{Enc}_K(N, A, M) = \mathrm{Enc}_{K'}(N', A', M')$, where $C = (C_0, C_1)$ and

$$\begin{aligned}
C_0 = C'_0 &= \{\mathtt{A2}, \mathtt{C6}, \mathtt{1F}, \mathtt{69}, \mathtt{67}, \mathtt{3C}, \mathtt{95}, \mathtt{8A}, \mathtt{86}, \mathtt{A1}, \mathtt{02}, \mathtt{0B}, \mathtt{8A}, \mathtt{6E}, \mathtt{B3}, \mathtt{BF}\},\\
C_1 = C'_1 &= \{\mathtt{B7}, \mathtt{2D}, \mathtt{CD}, \mathtt{79}, \mathtt{3D}, \mathtt{6A}, \mathtt{C2}, \mathtt{4D}, \mathtt{34}, \mathtt{0E}, \mathtt{33}, \mathtt{42}, \mathtt{31}, \mathtt{B1}, \mathtt{EA}, \mathtt{BB}\},\\
T = T' &= \{\mathtt{F0}, \mathtt{55}, \mathtt{36}, \mathtt{27}, \mathtt{C6}, \mathtt{63}, \mathtt{BC}, \mathtt{4D}, \mathtt{99}, \mathtt{B9}, \mathtt{5B}, \mathtt{54}, \mathtt{FF}, \mathtt{FF}, \mathtt{0E}, \mathtt{D8}\}.
\end{aligned}$$

Note that, to make $S_4[0, 1, 4..7] = S'_4[0, 1, 4..7]$ constant, we compute $A_2, \ldots, A_7$ as follows:

$$\begin{aligned}
A_7 &= \mathcal{A}(S_1[1] \oplus S_1[6]) \oplus \mathcal{A}(S_1[0]) \oplus S_1[7] \oplus S_4[4]\\
A_5 &= \mathcal{A}(S_1[2]) \oplus S_1[1] \oplus \mathcal{A}^{-1}(S_4[5] \oplus \mathcal{A}(S_1[1] \oplus S_1[6]) \oplus \mathcal{A}(S_1[0]) \oplus S_1[7])\\
A_3 &= S_1[3] \oplus \mathcal{A}^{-1}(\mathcal{A}(S_1[2]) \oplus S_1[1] \oplus \mathcal{A}^{-1}(S_4[6] \oplus \mathcal{A}(S_1[2]) \oplus S_1[1] \oplus A_5))\\
A_4 &= S_1[0] \oplus S_1[6] \oplus \mathcal{A}(\mathcal{A}(S_1[4]) \oplus S_1[3]) \oplus S_1[3] \oplus A_3 \oplus S_4[7]\\
A_2 &= S_4[1] \oplus \mathcal{A}(S_1[0] \oplus S_1[6] \oplus A_4) \oplus S_1[7] \oplus \mathcal{A}(S_1[5]) \oplus S_1[4]\\
A_6 &= S_1[7] \oplus A_2 \oplus \mathcal{A}(S_1[5]) \oplus S_1[4] \oplus S_4[0]
\end{aligned}$$