IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 3, pp. 123–151. DOI:10.46586/tosc.v2022.i3.123-151

Cryptanalysis of Rocca and Feasibility of Its Security Claim

Akinori Hosoyamada[1], Akiko Inoue[2], Ryoma Ito[3], Tetsu Iwata[4], Kazuhiko Mimematsu[2], Ferdinand Sibleyras[1] and Yosuke Todo[1]

[1] NTT Social Informatics Laboratories, Musashino, Japan
akinori.hosoyamada.bh@hco.ntt.co.jp,sibleyras.ferdinand.ez@hco.ntt.co.jp,yosuke.todo.xt@hco.ntt.co.jp
[2] NEC Corporation, Kawasaki, Japan
a_inoue@nec.com,k-minematsu@nec.com
[3] National Institute of Information and Communications Technology, Koganei, Japan
itorym@nict.go.jp
[4] Nagoya University, Nagoya, Japan
tetsu.iwata@nagoya-u.jp

Abstract. Rocca is an authenticated encryption with associated data scheme for beyond 5G/6G systems. It was proposed at FSE 2022/ToSC 2021(2), and the designers make a security claim of achieving 256-bit security against key-recovery and distinguishing attacks, and 128-bit security against forgery attacks (the security claim regarding distinguishing attacks was subsequently weakened in the full version in ePrint 2022/116). A notable aspect of the claim is the gap between the privacy and authenticity security. In particular, the security claim regarding key-recovery attacks allows an attacker to obtain multiple forgeries through the decryption oracle. In this paper, we first present a full key-recovery attack on Rocca. The data complexity of our attack is 2^{128} and the time complexity is about 2^{128}, where the attack makes use of the encryption and decryption oracles, and the success probability is almost 1. The attack recovers the entire 256-bit key in a single-key and nonce-respecting setting, breaking the 256-bit security claim against key-recovery attacks. We then extend the attack to various security models and discuss several countermeasures to see the feasibility of the security claim. Finally, we consider a theoretical question of whether achieving the security claim of Rocca is possible in the provable security paradigm. We present both negative and positive results to the question.

Keywords: AEAD · Rocca · Differential cryptanalysis · Releasing unverified plaintexts · Decryption oracle · IND-CCA

1 Introduction

Background. An authenticated encryption with associated data (AEAD) scheme is a symmetric key primitive to authenticate associated data (AD), and authenticate and encrypt messages. Various AEAD schemes have been proposed, and we focus on one of them called Rocca [SLN+21, SLN+22] that was proposed at FSE 2022/ToSC 2021(2). Rocca is an AES-based design that follows the design approach of AEGIS [WP13], Tiaoxin-346 [Nik14], and that of Jean and Nikolić [JN16]. In these designs, a round function is designed based on one AES round (aesenc) and a 128-bit XOR operation to fully take advantage of the AES-NI and SIMD (single instruction, multiple data) instructions.

Rocca was designed with the goal to meet the performance and security requirements in beyond 5G/6G systems. Concretely, it was designed to achieve 100 Gbps encryp-

Received: 2022-06-01 Accepted: 2022-08-01 Published: 2022-09-09

tion/decryption speed, 256-bit security against key-recovery attacks, and 128-bit security against forgery attacks. Indeed, [SLN+21] reports that Rocca achieves the speed of 138.22 Gbps on an Intel Ice-lake CPU, and the result shows that Rocca outperforms AEGIS and Tiaoxin-346, and other relevant schemes AES-256-GCM [MV04], ChaCha20-Poly1305 [NL05], and SNOW-V-GCM [EJMY19].

As for the security, the designers make the following claim in [SLN+21]:

Claim 1 ([SLN+21])**.** Rocca provides 256-bit security against key-recovery and distinguishing attacks and 128-bit security against forgery attacks in the nonce-respecting setting. We do not claim its security in the related-key and known-key settings.

We note that Rocca is a nonce-based AEAD scheme, meaning that its security relies on the uniqueness of the nonce, and we also note that the tag length of Rocca is 128 bits.

A notable aspect of Claim 1 is the gap between the privacy security and the authenticity security. In particular, the security claim regarding key-recovery attacks allows an attacker to obtain multiple forgeries through the decryption oracle. These observations bring us a natural question of feasibility and infeasibility of achieving the claim as we detail below. We remark that the security claim was weakened in [SLN+22] after the publication of [SLN+21]. Specifically, in [SLN+22], the indistinguishability security claim is weakened to 128-bit security, and various limitations on the input lengths are added[1]. To quote:

Claim 2 ([SLN+22])**.** Rocca provides 256-bit security against key-recovery and 128-bit security against distinguishing and forgery attacks in the nonce-respecting setting. We do not claim its security in the related-key and known-key settings.

The message length for a fixed key is limited to at most 2^{128} and we also limit the number of different messages that are produced for a fixed key to be at most 2^{128}. The length of associated data of a fixed key is up to 2^{64}.

In this paper, we present our security analysis of Rocca. We also present a theoretical analysis of the security claim focusing on Claim 1.

Key-Recovery Attacks on Rocca. We first present a key-recovery attack on Rocca. Our attack makes one encryption query and 2^{128} decryption queries, and recovers the entire 256-bit key with probability almost 1, where the time complexity is about 2^{128}. This attack is in a single-key setting and follows the nonce-respecting scenario, breaking the 256-bit security claim against key-recovery attacks made in both Claims 1 and 2.

As mentioned in the background, in Claims 1 and 2, 256-bit security regarding key-recovery attacks is claimed. However, the tag length is only 128 bits. These claims cannot invalidate any key-recovery attack exploiting multiple forgeries through a decryption oracle. It can be interpreted as, at the cost of 2^{128} decryption queries, an attacker is in the releasing unverified plaintext (RUP) setting [ABL+14], in which case the attacker has an oracle that returns a message of any ciphertext without verifying the correctness, and is free to repeat nonces.

In [SLN+21], the designers already observed the feasibility of a state-recovery attack under a nonce-misuse setting, however, the detailed attack procedure is not presented. In particular, it is interesting to see how many nonce-repeated plaintext-ciphertext pairs are required for the state-recovery attack. We show a detailed attack procedure exploiting the property of the AES S-box and recovering the entire 1024-bit state from only one nonce-repeated input-output pair. Moreover, we show a meet-in-the-middle technique that reduces the attack time complexity to practical. As a result, the time complexity is about 2^{20}, and the success probability is sufficiently high. Note that in Rocca, the state-recovery attack immediately leads to the key-recovery attack. Since the attack complexity is

[1] According to [SLN+22], the security claim was weakened for Grover's algorithm in the quantum setting.

practical (under the nonce-misuse/RUP setting), we implemented our key-recovery attack and verified the correctness[2].

Our attack is a chosen-ciphertext attack (CCA), and it is one of the critical attack scenarios for AEAD schemes as discussed, e.g., in [Mèg19, Kha22]. If the secret key of the AEAD is efficiently recovered under the RUP scenario as in Rocca, it is unlikely to enhance the security level beyond the tag length. Such an AEAD scheme is extremely vulnerable when the tag is truncated. For example, our attack implies that if the tag length of Rocca is reduced to 32 bits, it only ensures 32-bit security against all kinds of attacks. It is instructive to consider the impact of CCAs to understand the risk of truncating the tag.

We next discuss extensions of the above attack to various other security models. In the above attack, the attacker has the encryption and decryption oracles. We point out that limiting the number of decryption queries still gives a key-recovery attack that is faster than the exhaustive key search. We also consider the case where the attacker has the decryption oracle only, and the case where the nonce-respecting condition is applied to the decryption oracle as well. The latter case is highly impractical, while a key-recovery attack is still possible for both cases.

Then, we consider several approaches of countermeasures. This includes increasing the number of rounds in the initialization and/or finalization, increasing the nonce length, and increasing the tag length, and we conclude that none of them works. The most promising idea (with negligible impact on the cost) to reduce the impact of our attack is to use the secret key after the initialization and before the finalization as is done, e.g., in ASCON [DEMS21].

Theoretical Analysis of the Security Claim. We next turn our attention to a theoretical analysis of Claim 1 of Rocca. Specifically, our attention is on the gap in the bit security between the distinguishing and forgery attacks. We present theoretical analyses that are valid not only for Rocca but also for any AEAD with different bit security for distinguishing and forgery attacks.

We observe that our key-recovery attack above also invalidates the 256-bit security claim against distinguishing attacks in Claim 1, depending on the interpretation of the security model[3]. As mentioned, Rocca is a nonce-based AEAD scheme and its security relies on the uniqueness of the nonce. However, since the tag length of Rocca is 128 bits, it is always possible to obtain a nonce-repeated input-output pair after 2^{128} decryption queries, and whether the attacker has the decryption oracle (in a distinguishing attack) depends on the security model considered. It is often the case that the security against distinguishing attacks is modelled as the indistinguishability against chosen-plaintext attacks (IND-CPA), and the security against forgery attacks is modelled as the integrity of ciphertexts (INT-CTXT), as these two notions imply the indistinguishability against CCAs (IND-CCA) and also imply the unified AEAD security notion, which are regarded as the right security notions for AEAD schemes to achieve [BN08, NRS14, RS06, NRS13]. Indeed, a significant number of AEAD schemes with a proof of security aim at proving the IND-CPA security and INT-CTXT security. See, e.g., [Rog04a, KR11, IOM12].

Now the entire security as AEAD in the IND-CCA notion or in the unified AEAD notion is given as the lower bound of the two notions, i.e., if an AEAD scheme has k_1-bit IND-CPA security and k_2-bit INT-CTXT security, then it ensures $\min\{k_1, k_2\}$-bit IND-CCA security and $\min\{k_1, k_2\}$-bit AEAD security (in the unified notion). See [Mèg19, Kha22] for a related discussion. In many cases, the security suggested by the IND-CPA bound and

[2]The source code of our attack is available at the following URL: https://github.com/Sibleyras/KeyRecoveryNonceMisusedRocca

[3]This does not break the 128-bit security claim against distinguishing attacks in Claim 2. The claim was weakened for Grover's algorithm in the quantum setting, which is irrelevant to the analysis of this paper and we focus on Claim 1.

INT-CTXT bound are comparable, and this approach does not impose visible impact on the security of the AEAD scheme as a whole. However, this is not the case in Claim 1.

Claim 1 can be interpreted at least in two ways. One is to achieve 256-bit IND-CPA security and 128-bit INT-CTXT security. This is motivated by the approach taken by many AEAD schemes, where the difference is that Rocca has a gap between the two, and if we follow the discussion above, it ensures only 128-bit IND-CCA security. As the limitation of having a stronger IND-CPA security claim, this does not cause an issue if real world attackers are isolated from a decryption oracle, in which case users can benefit from the strong IND-CPA security bound. However, in this case, it is not clear whether using an AEAD scheme is needed in the first place, as the functionality of authenticity is redundant if the adversary does not have the decryption oracle. In contrast to this, in many use cases of AEAD schemes, real world attackers do have access to the decryption oracle and users expect that the security is maintained in this situation, which is one of the primal reasons to use an AEAD scheme.

In order to capture this, another way to interpret Claim 1 is to achieve 256-bit IND-CCA security and 128-bit INT-CTXT security, and the question we ask is the feasibility and infeasibility of achieving this type of security, together with the 256-bit security against key-recovery attacks, in the provable security paradigm, where we consider schemes with 128-bit tags as in Rocca.

Feasibility and Infeasibility of the Security Claim. In order to give the answer to the question, we start by pointing out that achieving 256-bit IND-CPA security and 128-bit INT-CTXT security is possible with known approaches. Concretely, we point out that a variant of GCM [MV04], OCB [KR11], OPP [GJMN16], and duplex sponge [BDPA11] can achieve the security. However, as stated above, this does not necessarily imply that 256-bit privacy is guaranteed in an environment where attackers have a decryption oracle.

We next show that a class of AEAD schemes called an online AEAD scheme [BBKN12] cannot achieve 256-bit IND-CCA security, by presenting a distinguishing attack with 2^{128} query complexity. In an online AEAD scheme, the i-th output block depends only on the first i blocks of input, and this class includes all the schemes stated above and Rocca. We remark that an online AEAD scheme here is different from those studied in [FFL12, HRRV15] in that the goal is to have the best possible security under nonce repeating scenario.

This result rules out efficient solutions, and we present our feasibility result to answer the question above with an offline construction. We show that the Encode-then-Encipher approach [BR00] with an appropriate assumption on the interface can simultaneously achieve 256-bit IND-CCA security and 128-bit INT-CTXT security. The efficiency is limited, while this result does show that achieving Claim 1 in a provable security paradigm is feasible. We remark that works on AEAD schemes with variable stretches [RVV16, GRV21] consider a problem of varying the tag length during the lifetime of the key, which is a different problem from the focus of this paper.

Organization. In Sect. 2, we review the specification of Rocca. In Sect. 3, we present our key-recovery attack and discuss extensions of the attack to several security models. In Sect. 4, we consider countermeasures to mitigate the impact of our attack. In Sect. 5, we present our theoretical treatment of achieving Claim 1. We conclude the paper in Sect. 6.

2 Specification of Rocca

Rocca is an authenticated encryption with associated data (AEAD) scheme. Rocca has a $128 \times 8 = 1024$-bit internal state, and the state is updated by the round update function

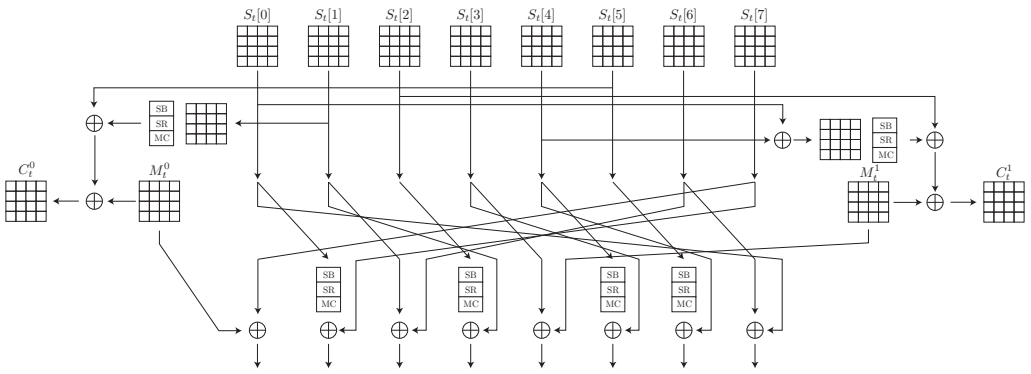

Figure 1: The update function of Rocca.

R while absorbing associated data and a message, and squeezing output to encrypt the message.

2.1 Notation

We use the following notations in the paper.

- S: The state of Rocca, which is composed of 8 blocks, i.e., $S = (S[0], S[1], \ldots, S[7])$, where for $0 \leq i \leq 7$, $S[i]$ is a 128-bit string, and $S[0]$ is the first block. When we focus on the state at time t, we use $S_t = (S_t[0], S_t[1], \ldots, S_t[7])$.

- Z^0: A 128-bit constant block defined as $Z^0 =$ `428a2f98d728ae227137449123ef65cd` (in hex).

- Z^1: A 128-bit constant block defined as $Z^1 =$ `b5c0fbcfec4d3b2fe9b5dba58189dbbc` (in hex).

- $A(X)$: The AES round function without AddRoundKey, as defined below:

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

  where MixColumns, ShiftRows and SubBytes are the same operations as defined in AES.

- $R(S_t, X^0, X^1)$: The round function used to update the state S_t.

For a bit string X, $|X|$ denotes the the length of X in bits. We write 0^l for a zero string of length l bits. For bit strings X and Y, $X \,\|\, Y$ denotes their concatenation.

2.2 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, X^1). The output $S_{t+1} \leftarrow R(S_t, X^0, X^1)$ is computed as follows:

$$\begin{aligned}
S_{t+1}[0] &= S_t[7] \oplus X^0\,,\\
S_{t+1}[1] &= A(S_t[0]) \oplus S_t[7]\,,\\
S_{t+1}[2] &= S_t[1] \oplus S_t[6]\,,\\
S_{t+1}[3] &= A(S_t[2]) \oplus S_t[1]\,,\\
S_{t+1}[4] &= S_t[3] \oplus X^1\,,\\
S_{t+1}[5] &= A(S_t[4]) \oplus S_t[3]\,,\\
S_{t+1}[6] &= A(S_t[5]) \oplus S_t[4]\,,\\
S_{t+1}[7] &= S_t[0] \oplus S_t[6]\,.
\end{aligned}$$

Figure 1 shows the update function, where $X^0 = M_t^0$ and $X^1 = M_t^1$.

2.3 The Mode of Operation of Rocca

The processing of Rocca consists of four phases: initialization, processing associated data, encryption, and finalization. Rocca accepts a 256-bit key $K_0 \,\|\, K_1 \in \mathbb{F}_2^{128} \times \mathbb{F}_2^{128}$, a 128-bit nonce N, associated data AD, and a message M as input. The output is the corresponding ciphertext C and a 128-bit tag T. For a string X of any bit length, define $\overline{X} = X \,\|\, 0^l$, where l is the minimal non-negative integer such that $|\overline{X}|$ is a multiple of 256. In addition, for a string X of $|X|$ being a multiple of 256, we write X as $X = X_0 \,\|\, X_1 \,\|\, \ldots \,\|\, X_{\frac{|X|}{256}-1}$ with $|X_i| = 256$. Further, X_i is written as $X_i = X_i^0 \,\|\, X_i^1$ with $|X_i^0| = |X_i^1| = 128$.

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

$$(S[0], \ldots, S[7]) = (K_1, N, Z^0, Z^1, N \oplus K_1, 0, K_0, 0)\,.$$

Then, 20 iterations of the round update function $R(S, Z^0, Z^1)$ is applied to the state.

Processing the Associated Data. Associated data AD is padded to $\overline{AD}$ and the state is updated as follows:

$$S_{t+1} = R(S_t, \overline{AD}_t^0, \overline{AD}_t^1)$$

until $t = \frac{|\overline{AD}|}{256}$. Note that this phase is skipped if AD is empty.

Processing Message. On encryption, we process a message as follows: A message M is first padded to $\overline{M}$. Then, $\overline{M}$ is absorbed with the round function, and the corresponding ciphertext C is generated. If the length of the last block of M is b bits for some $0 < b < 256$, the last block of C is truncated to the first b bits. A detailed procedure is shown as follows:

$$\begin{aligned}
C_t^0 &= A(S_t[1]) \oplus S_t[5] \oplus \overline{M}_t^0\,,\\
C_t^1 &= A(S_t[0] \oplus S_t[4]) \oplus S_t[2] \oplus \overline{M}_t^1\,,\\
S_{t+1} &= R(S_t, \overline{M}_t^0, \overline{M}_t^1)\,.
\end{aligned}$$

Note that this phase is skipped if M is empty.

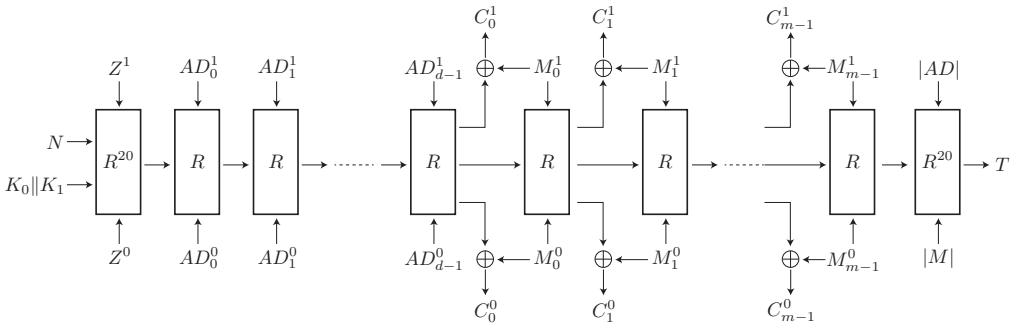

Figure 2: The encryption procedure of Rocca.

Processing Ciphertext. On decryption, we process a ciphertext as follows: A ciphertext C is first padded to $\overline{C}$. Then, $\overline{C}$ is absorbed with the round function, and the corresponding message M is generated. If the length of the last block of C is b bits for some $0 < b < 256$, the last block of M is truncated to the first b bits. A detailed procedure is shown as follows:

$$\begin{aligned} M_t^0 &= A(S_t[1]) \oplus S_t[5] \oplus \overline{C}_t^0\,, \\ M_t^1 &= A(S_t[0] \oplus S_t[4]) \oplus S_t[2] \oplus \overline{C}_t^1\,, \\ S_{t+1} &= R(S_t, \overline{M}_t^0, \overline{M}_t^1)\,. \end{aligned}$$

Note that this phase is skipped if C is empty.

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

$$T = \sum_{i=0}^{7} S[i]\,.$$

On decryption, the computed tag is compared with the tag given as input. Figure 2 shows the encryption procedure of Rocca.

3 Key-Recovery Attacks on Rocca

3.1 Attack Concept

We first review the security claims of Rocca from Claims 1 and 2. Rocca outputs a 128-bit tag and claims 128-bit security against forgery attacks and 256-bit security against key-recovery attacks. Rocca is a nonce-based AEAD scheme. Namely, attackers cannot ask for different messages with the same nonce to the encryption oracle.

These security claims are sound at first glance. However, when a decryption oracle is available to an attacker, the decryption oracle returns a valid plaintext-ciphertext pair when the 128-bit tag is consistent. Therefore, "nonce-repeated" pairs can be collected even if the attacker follows the nonce-respecting scenario with respect to the encryption oracle. This observation leads to the following attack procedure.

1. The attacker makes an encryption query (N, M) to obtain (C, T), where N is any nonce and M is any message whose length is properly chosen (specifically, at least eight blocks for our key-recovery attacks).

2. The attacker injects a proper difference Δ to the ciphertext and makes a decryption query $(N, C \oplus \Delta, T')$ while trying out q_d possible values of T'. The procedure returns a "nonce-repeated" plaintext (N, M') with a probability of $\frac{q_d}{2^{128}}$.

3. The attacker recovers the internal state by exploiting the collected "nonce-repeated" pair. The attacker recovers the secret key from the recovered internal state by applying the inverse of the round update function.

The procedure above shows that attackers can collect nonce-repeated pairs even in the nonce-respecting scenario. In [SLN+21], the designers noted that the state-recovery attack seems trivial if the nonce is misused. That is, when nonce-repeated pairs are given, the state-recovery attack would be possible.

It remains to determine the number of "nonce-repeated" pairs needed to recover the state. In the concrete attack against Rocca (shown in the following subsection), we show the attack procedure to recover the internal state by using only one "nonce-repeated" pair with a practical time complexity. We combine the state-recovery attack using a "nonce-repeated" pair with the attack exploiting the decryption oracle. With a sufficiently large number of decryption queries, q_d, we obtain a key-recovery attack with data complexity q_d that recovers the 256-bit secret key with a probability of $\frac{q_d}{2^{128}}$ under the nonce-respecting scenario. When $q_d = 2^{128}$, it recovers the 256-bit secret key with a probability of 1 and a time complexity of about 2^{128}.

Although each attack is not surprising, the combined attack breaks the claimed security. In practice, there is a demand to truncate the tag length on a real security system. Users may expect that the tag truncation does not affect the claimed security except for the security against forgery attacks. However, our attack implies that, in Rocca, the tag truncation directly degrades all the security claims, e.g., Rocca with a 32-bit tag ensures only 32-bit security.

3.2 Key-Recovery Attack using A Nonce-Repeated Pair

As noted by the designers in [SLN+21], the state-recovery attack would be trivial when a nonce is misused. However, it is still challenging if we can recover the secret key when only one nonce-repeated pair is available. We tackle this problem, and surprisingly, we show only one nonce-repeated pair is enough to recover the whole 1024-bit internal state with the practical time complexity, i.e., about 2^{20}. Note that the state recovery of Rocca directly leads to the secret key recovery.

We describe our attack under the releasing unverified plaintext (RUP) setting because of a straightforward conversion to the attack exploiting the decryption oracle in the nonce-respecting scenario. Note that the same attack works when the nonce is misused in the encryption oracle.

3.2.1 State-Recovery Attack

We first introduce a well-known property regarding input-output differences through the AES S-box, which is another view of Observation 1 in [BDD+12].

Lemma 1. *Let x be an unknown random input of the AES S-box. Given the knowledge of a non-zero input difference Δ_{in}, there are 128 distinct pairs of input differences, $(x, x \oplus \Delta_{in})$. We split these pairs into subsets such that any pair in the same subset implies the same $S(x) \oplus S(x \oplus \Delta_{in}) = \Delta_{out}$. Then, independently of Δ_{in}, we have 63 subsets: one subset contains four pairs, and 62 subsets contain two pairs. Thus, the observed output difference*

Δ_{out} implies two candidates of x with a probability of $124/128 = 31/32$ and four candidates with a probability of $1/32$.

Given an input-output difference of the AES S-box, we can reduce the number of candidates of corresponding input-output values to two in many cases. With a probability of $1/32$, the number of candidates is four. They correspond to every entry of the differential distribution table of the AES S-box.

We next show a concrete attack procedure to recover the internal state. Figure 3 shows the differential transition on the decryption process when non-zero byte differences are injected to 16 bytes of C_0^1, where SB, SR, and MC denote SubBytes, ShiftRows, and MixColumns, respectively. We call this a proper difference. The message needs to contain at least eight blocks to mount our state-recovery attack. White-colored bytes are constant, yellow-colored bytes have known differences, and gray-colored bytes have unknown differences. The green-colored number shows the order of the 128-bit state whose candidates are narrowed down following the step number. Moreover, for the sake of simplicity of the attack procedure, we assume the case that each yellow-colored byte has a non-zero difference. With a low probability, a few yellow-colored bytes do not have differences due to a collision. Then, a complicated procedure is required, and we discuss such a case later.

Step 1. We focus on the round function labeled with A. The input difference is equal to ΔC_0^1. The output difference is equal to ΔM_1^1. Due to Lemma 1, the number of candidates of the input, $S_1[4] \oplus S_1[0]$, is reduced to 2^{16+i_1} with a probability of $\binom{16}{i_1} \times (1/32)^{i_1} \times (31/32)^{16-i_1}$. These probabilities are about 0.6, 0.31, 0.075, and 0.011 for $i_1 = 0$, $i_1 = 1$, $i_1 = 2$, and $i_1 = 3$, respectively. Thus, the probability that the number of candidates is narrowed down to 2^{16} is the highest, and the probability is higher than 99% for $i_1 \leq 3$. Note that $S_1[2]$ is computed as $A(S_1[4] \oplus S_1[0]) \oplus (M_1^1 \oplus C_1^1)$.

Step 2. We focus on the round function labeled with B. The input difference is equal to ΔC_0^1. The output difference is equal to ΔM_2^0. Due to Lemma 1 and similarly to Step 1, the number of candidates of the input, $S_1[4]$, is reduced to 2^{16+i_2} with a high probability for small i_2. In Step 1, each byte of $S_1[4] \oplus S_1[0]$ has two candidates and let a and $a \oplus \delta$ be the candidates. In Step 2, each byte of $S_1[4]$ also has two candidates and let b and $b \oplus \delta$ be the candidates. Remark that the common difference δ appears in Step 1 and Step 2. Thus, each byte of $S_1[0]$ has two candidate, $a \oplus b$ or $a \oplus b \oplus \delta$. Therefore, the number of candidates of $S_1[0]$ is reduced to $2^{16+i_1+i_2}$.

Step 3. We focus on the round function labeled with C. Similarly to Step 1, the number of candidates of the input, $S_2[4] \oplus S_2[0]$, is reduced to 2^{16+i_3} for small i_3, and $S_2[2]$ is computed.

Step 4. We focus on the round function labeled with D. Similarly to Step 2, the number of candidates of the input, $S_2[4]$, is reduced to 2^{16+i_4} for small i_4. The number of candidates of $S_2[0]$ is also reduced to $2^{16+i_3+i_4}$.

Step 5. On each guess, we recover the following state blocks:

- $S_1[3] = S_2[4] \oplus M_1^1$
- $S_2[5] = A(S_1[4]) \oplus S_1[3]$
- $S_2[1] = A^{-1}(M_2^0 \oplus C_2^0 \oplus S_2[5])$
- $S_1[7] = A(S_1[0]) \oplus S_2[1]$

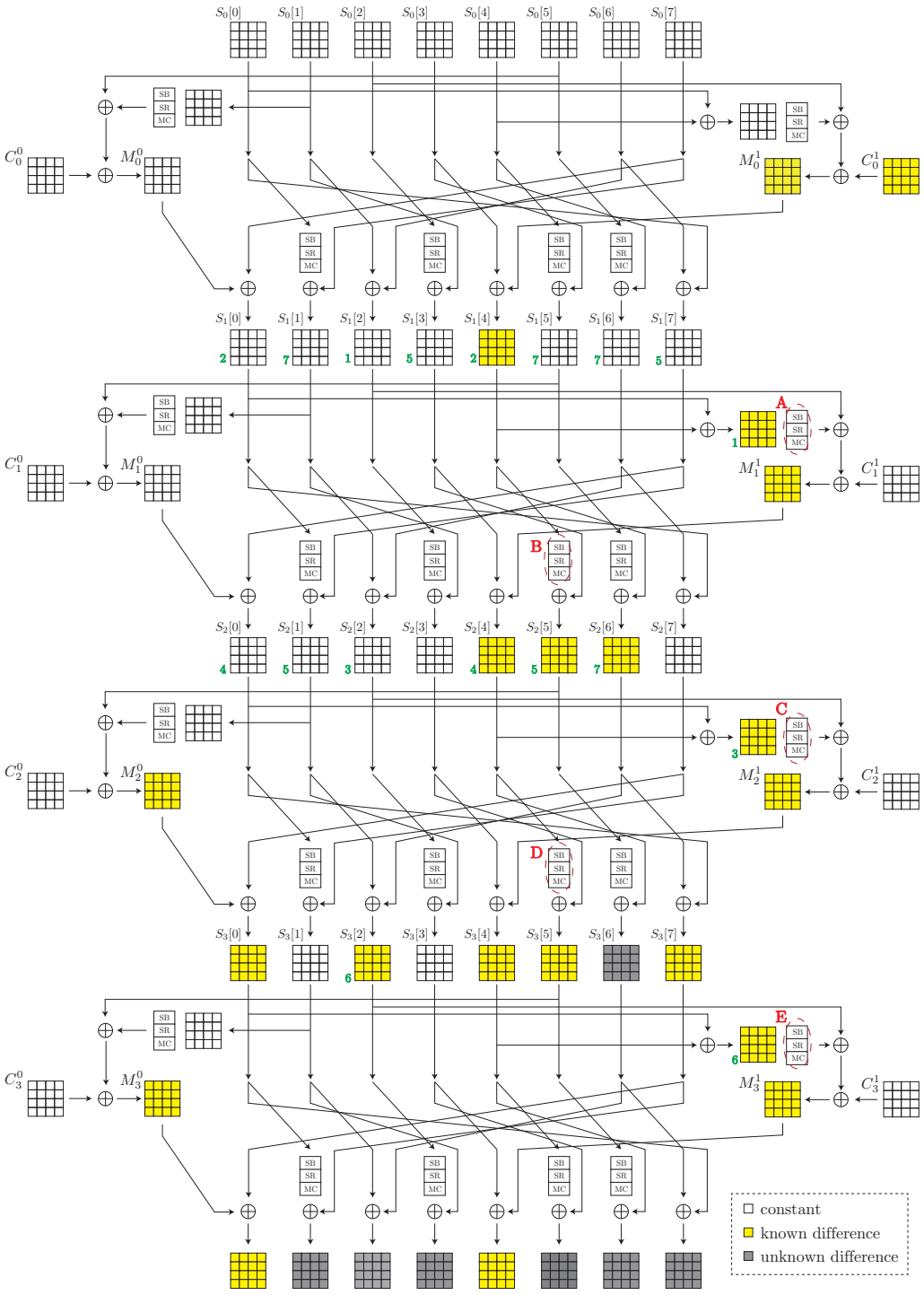

Figure 3: State recovery using one “nonce-repeated” pair.

Then, we check if

$$M_1^0 = S_1[7] \oplus S_2[0] \tag{1}$$

holds. The total number of candidates is $2^{16+i_4} \times 2^{16+i_2} \times 2^{16+i_1+i_2} \times 2^{16+i_3+i_4} = 2^{64+i_1+2i_2+i_3+2i_4}$. Note that $i_1 + 2i_2 + i_3 + 2i_4$ is small and Eq.(1) holds with a probability of 2^{-128}. Therefore, we can identify the correct one from the $2^{64+i_1+2i_2+i_3+2i_4}$ candidates[4].

Step 6. We focus on the round function labeled with E. The number of candidates of the input, $S_3[4] \oplus S_3[0]$, is reduced to 2^{16+i_6} for small i_6, and $S_3[2]$ is computed.

Step 7. On each guess, we recover the following state blocks:

- $S_2[6] = S_2[1] \oplus S_3[2]$
- $S_1[5] = A^{-1}(S_1[4] \oplus S_2[6])$
- $S_1[1] = A^{-1}(M_1^0 \oplus C_1^0 \oplus S_1[5])$
- $S_1[6] = S_2[2] \oplus S_1[1]$

Then, the number of candidates of the whole state of S_1 is 2^{16+i_6}, and the unique one is easily recovered by observing recovered plaintext blocks or the tag. Once the whole state is recovered, we immediately recover the secret key.

3.2.2 Reducing Complexity Using Meet-in-the-Middle Technique

The attack described in Sect. 3.2.1 requires only one "nonce-repeated" pair and at least 2^{64} time complexity. Since the required complexity is significantly lower than 2^{256}, the attack already breaks the claimed security by combining the attack exploiting the decryption oracle. However, it is not a practical attack.

The dominant step is Step 5. We can reduce the attack complexity significantly to be practical thanks to the meet-in-the-middle technique. For the sake of simplicity, we describe the case where $i_1 = i_2 = i_3 = i_4 = 0$. The complexity is slightly larger but never significantly larger when they are not zero. We discuss this case in detail in Sect. 3.2.4.

Figure 4 shows the meet-in-the-middle relation, where red line and blue line can be computed independently. Two lines collide in each column of $S_2[5]$. In the red line, $S_2[5]$ is computed as

$$S_2[5] = A(A(S_1[0]) \oplus M_1^0 \oplus S_2[0]) \oplus C_2^0 \oplus M_2^0 \,.$$

When each column of $S_2[5]$ is computed, the whole state of $S_1[0]$ is involved but only one diagonal of $S_2[0]$ is involved. Therefore, there are $2^{16} \times 2^4 = 2^{20}$ candidates of each column of $S_2[5]$. On the other hand, in the blue line, $S_2[5]$ is computed as

$$S_2[5] = A(S_1[4]) \oplus S_2[4] \oplus M_1^1 \,.$$

When each column of $S_2[5]$ is computed, one diagonal of $S_1[4]$ and one column of $S_2[4]$ are involved. Therefore, there are $2^4 \times 2^4 = 2^8$ candidates of each column of $S_2[5]$. On the meet-in-the-middle process, there are $2^{20} \times 2^8 = 2^{28}$ candidates, and they are filtered by a 32-bit collision that happens with a probability of 2^{-32} at random. Therefore, by changing target columns, we recover the unique solution from 2^{64} candidates with the complexity of 2^{20} only.

[4]In practice, about 10% time we get a few additional solutions that only slightly differ from the correct one. Specifically, let $\tilde{S}_2[1]$ and $\tilde{S}_1[4]$ be incorrect guesses that only differ from the real value on a single byte at the same position. This always exists as each input byte of C and B are guessed independently. Therefore $S(S_2[1]) \oplus S(\tilde{S}_2[1]) \oplus S(S_1[4]) \oplus S(\tilde{S}_1[4]) = 0$ is actually an 8-bit condition that passes with probability 1/256. Hence with about $(16 + i_3)/256$ probability there are at least 1 byte that pass the condition and we have multiple solutions. It is unlikely that we have many solutions. Thus, we go to Step 6 for each guess and get the unique one eventually.

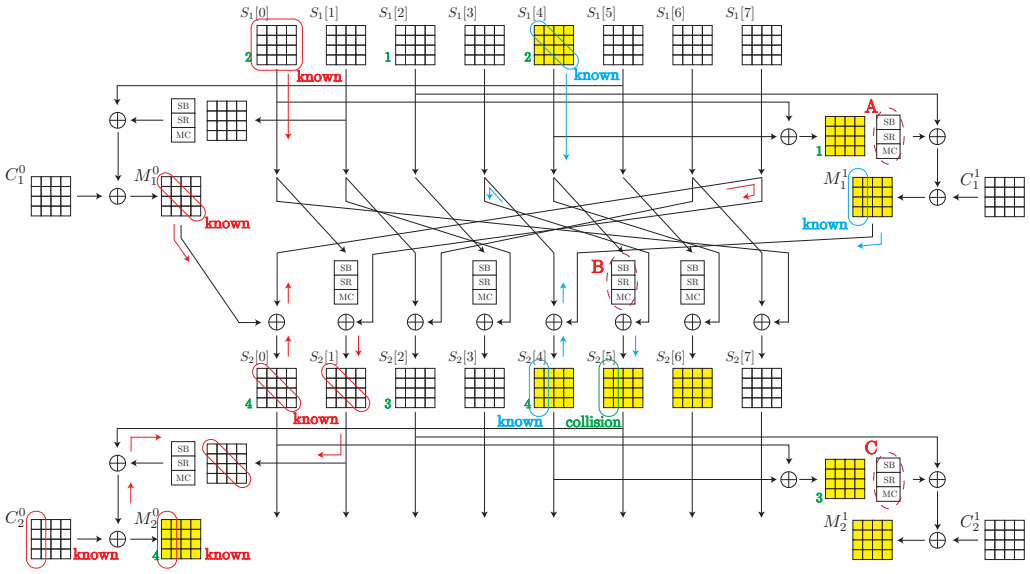

Figure 4: Meet-in-the-middle technique.

3.2.3 Case of Collisions in Some Bytes with Known Difference

The procedure above assumes the case that all yellow-colored bytes have a non-zero difference. Recall Lemma 1. If the corresponding byte difference is zero by chance, we cannot narrow down the number of candidates.

See Figure 3. Lemma 1 is applied to five AES round functions labeled with A, B, C, D, and E. Input differences in AES round functions labeled with A and B coincide with ΔC_0^1, which is chosen by an attacker. Therefore, we can avoid collisions happening here by injecting a proper difference. Unfortunately, input differences in AES round functions labeled with C, D, and E are uncontrollable. Since C and D take the same input difference, the probability that we do not observe any collision is $(255/256)^{32} \approx 88.2\%$.

We now discuss the case that there are some colliding bytes by chance. Then, 2^8 candidates remain in the corresponding bytes instead of 2 or 4 candidates.

We recall the meet-in-the-middle procedure, where 16 bytes of $S_1[0]$ and 4 bytes of $S_2[0]$ are used in the red line, and 4 bytes of $S_1[4]$ and 4 bytes of $S_2[4]$ are used in the blue line. Note that all bytes in $S_1[0]$ and $S_1[4]$ are narrowed down thanks to the proper choice of ΔC_0^1. We can choose the order of target colliding columns in $S_2[5]$ such that the increase of the time complexity is as low as possible. Namely, assuming that the first byte of $S_2[0]$ has 2^8 candidates and the others have at most 4 candidates, we first target a non-first column in $S_2[5]$, determine $S_1[0]$ first, and then target the first column. Then, we can save the increase of the attack complexity.

We observe some colliding bytes in E too. When there are at most x colliding bytes out of 16 bytes, the probability is $\sum_{i=0}^{x} \binom{16}{i}(1/256)^i(255/256)^{16-i}$, and the probability is almost 1 when $x \leq 2$. Even if we have 2 such bytes, the complexity of Step 6 is $2^{16+i_6-2+16} = 2^{30+i_6}$, which is still practical.

Note that this unfortunate case only happens in the RUP setting. On the nonce-misuse setting, we can avoid it by choosing message differences such that yellow-colored bytes have non-zero differences.

Table 1: Relationship between the time complexity and success probability for the proposed attack, where "time" denotes the number of candidates in our state-recovery attack. This result was obtained by conducting experiments with 2^{20} trials.

Time	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}	2^{26}	2^{27}	2^{28}	2^{29}	2^{30}	2^{31}
Success prob.	0.565	0.854	0.924	0.968	0.989	0.996	0.997	0.997	0.997	0.998	0.999	0.999

3.2.4 Experimental Verification

We conducted experiments to verify the validity of the proposed attack. The following is our experimental environment: a Linux machine with 28-core Intel(R) Xeon(R) Gold 6258R CPU (2.70 GHz), 256.0 GB of main memory, a gcc 9.4.0 compiler, and the C programming language. We used the Mersenne Twister[5], which is a pseudorandom number generator proposed by Matsumoto and Nishimura [MN98], to generate the secret keys, nonces, associated data, and messages used in all our experiments, and thus did not reuse them in any of the experiments. Our experimental verification procedure is as follows:

Step 1. We generate a secret key K $(= K_0 \| K_1)$, a nonce N, associated data AD, and a message M at random, and then simulate the encryption oracle to get the ciphertext C from a tuple (K, N, AD, M).

Step 2. We inject a proper difference Δ to the ciphertext C, and then simulate the decryption oracle to get the message M' from a tuple $(K, N, AD, C \oplus \Delta)$. We used $\Delta C_0^1 =$ `0x01010101010101010101010101010101` in our experiments. Note that we use the decryption oracle that releases unverified plaintexts to make the experiment practical.

Step 3. We use the "nonce-repeated" pair (M, C) and $(M', C \oplus \Delta)$ to recover the whole internal state of S_1 based on the proposed attack procedure described in Sects. 3.2.1 and 3.2.2.

We provide a test case for the proposed attack in Appendix A. After completing multiple trials with the above steps, we estimate the time complexity and success probability for the proposed attack while taking the specific case explained in Sect. 3.2.3 into consideration. To this end, we conducted experiments with 2^{20} trials, i.e., we used 2^{20} different tuples (K, N, AD, M) to verify the proposed attack.

Table 1 summarizes the results of estimating the time complexity and success probability for the proposed attack. As described in the beginning of Sect. 3.2.1, given an input-output difference of the AES S-box, we can reduce the number of candidates of corresponding input-output values to two with a probability of 31/32, whereas the number of candidates is four with a probability of 1/32. This suggests that the proposed attack can be optimized (i.e., the proposed attack can be performed with a time complexity of 2^{20}) only when all of our guesses can be narrowed down to two candidates. However, it is not always possible to optimize. Indeed, Table 1 indicates that the proposed attack can be performed with an optimized time complexity of 2^{20}, but its success probability is about 0.565.

There are three cases in which the attack complexity increases from 2^{20}: The number of candidates for given input-output differences is four, an input difference of the target S-boxes becomes zero by chance, and an incorrect guess passes through a 128-bit filter. Taking all these cases into consideration, we summarize the success probability of our attack with time complexity $2^{20} \times \alpha$ for $\alpha \leq 2^{11}$ by 2^{20} experiments in Table 1. We observe that the success probability is sufficiently high for all the values of α we considered. Therefore, our experimental verification demonstrates that the proposed attack is practical.

[5]The source code is available at https://github.com/omitakahiro/omitakahiro.github.io/blob/master/random/code/MT.h

Table 2: Summary of attacks under various security models. q_{enc} denotes the type and the number of encryption queries, where KPA denotes a known-plaintext query and CPA denotes a chosen-plaintext query. q_{dec} denotes the type and the number of decryption queries. α denotes a small constant when the attack complexity increases slightly.

Model	q_{enc}	q_{dec}	Time	Success prob.	Nonce respect
nonce misuse	1 KPA + 1 CPA	-	$\alpha \cdot 2^{20}$	≈ 1	✗
enc and dec	1 KPA	2^{128} CCA	$2^{128} + \alpha \cdot 2^{20}$	≈ 1	✓
dec query limit	1 KPA	q_d CCA	$q_d + \alpha \cdot 2^{20}$	$q_d/2^{128}$	✓
no enc	-	2^{129} CCA	$2^{129} + \alpha \cdot 2^{20}$	≈ 1	✓
dec w/o nonce repeat	q KPA	q CCA	$2q + \alpha \cdot 2^{20}$	$q/2^{128}$	✓

3.2.5 Summary of Attacks

Once an attacker obtains one "nonce-repeated" pair (with a proper difference), it is possible to recover the secret key with the complexity of about 2^{20}. The attack implies a risk of immediate recovery of the secret key if a nonce is misused even only once. Besides, we can convert the nonce-misusing attack with the attack exploiting the decryption oracle. That is, an attacker makes a known-plaintext query to the encryption oracle, injects a proper difference to the ciphertext, and queries the modified ciphertexts to the decryption oracle while trying out all the 2^{128} possible tags. Since 2^{128} trials always return the corresponding plaintexts, the attacker can recover the secret key with a success probability of 1 and the time complexity of $2^{128} + 2^{20}$. This attack is more efficient than an exhaustive search of the 256-bit key.

3.3 Extension to Various Security Models

In addition to the attack with one encryption query and 2^{128} decryption queries, in this section, we discuss the feasibility of the attack in three specific security models: 1) The case where the number of decryption queries that an attacker can make is limited. 2) The case where the encryption oracle is unavailable to the attacker. 3) The case where the decryption oracle is stateful and does not allow nonce-repeated decryption queries. Interestingly, in all these security models, Rocca is insecure. Table 2 shows the summary of attacks against Rocca in these security models.

Query Limitation. Our attack shown in Sect. 3.1 requires 2^{128} decryption queries to collect a nonce-repeated pair. When the number of decryption queries q_d is limited, e.g., $q_d \ll 2^{128}$, the attacker cannot query over 2^{128} tags to the decryption oracle. In this case, the success probability of our attack decreases from 1 to $q_d/2^{128}$, and the time complexity of the attack is $q_d + 2^{20}$. Although the success probability never reaches 1, the attack is still more efficient than the exhaustive search with the same success probability. For example, when $q_d = 2^{64}$, the time complexity is about 2^{64} with the success probability of 2^{-64}. The corresponding exhaustive search with the same success probability would require the time complexity of $2^{256-64} = 2^{192}$.

No Encryption Oracle. Interestingly, there is a variant of the attack that no longer requires an encryption oracle. An attacker makes decryption queries with a random ciphertext while trying out 2^{128} possible tags and obtains the corresponding message. Then, the attacker injects a proper difference to the ciphertext and runs the same attack. This variant requires 2^{129} decryption queries, and the time complexity is $2^{129} + 2^{20}$. This variant no longer requires the encryption oracle.

Decryption with Nonce-Repeat Detection. Let us assume a decryption oracle with a state to reject ciphertexts whose nonce is already used in a previous decryption query. Such a decryption oracle is not common because it is hard to implement. Besides, when a nonce is rejected although an already-queried ciphertext with the same nonce was not valid on the decryption, attackers can easily exhaust nonces, and the DoS attack works. Even such an impractical and complicated decryption oracle cannot invalidate our attack. Our attack can recover the secret key with only one nonce-repeated pair. Now an attacker makes an encryption query and obtains a plaintext-ciphertext pair. Then, the attacker injects a difference to the ciphertext and makes a decryption with the same nonce used in the encryption query. If it is rejected, we no longer use this nonce for both encryption and decryption queries. Instead, the attacker next makes an encryption query with a fresh nonce and repeats this until the decryption oracle returns a valid message. With q encryption and decryption queries, the time complexity of this attack is $2q + 2^{20}$ with the success probability of $q/2^{128}$.

4 Countermeasures

In this section, we consider several approaches to tweak Rocca to mitigate the impact of our attack.

4.1 Parameter Change

We first discuss if using different parameters, the number of rounds in the initialization and finalization, the nonce length, and the tag length, can invalidate the attack.

Our attack does not exploit the concrete structure of the initialization and finalization. Even if the initialization and finalization have more rounds, such a tweak never invalidates our attack.

Our attack works in the nonce-respecting setting and requires only one encryption query. Therefore, increasing the nonce length does not invalidate our attack.

While increasing the tag length is relatively promising, we need to evaluate whether Rocca ensures the higher security corresponding to the tag length against forgery attacks. Let us consider the case that we increase the tag length from 128 bits to 256 bits. The designers guarantee at least 24 active S-boxes in differential trails available in the forgery attack. It is insufficient to ensure 256-bit security against the forgery attack. We evaluated the tight maximum differential characteristic probability to confirm the insufficiency. As a result, we found a differential trail available in the forgery attack with a probability of $2^{-6\times 25} = 2^{-150}$. The existence of this differential trail implies that increasing the tag length is not a promising direction to invalidate the attack using the decryption oracle. We refer to Appendix B for details.

4.2 Key-Dependent Initialization and Finalization

To invalidate our attack, the most promising idea (with negligible impact on the cost) is involving the secret key to the initialization and finalization. Such an idea was already introduced in ASCON to enhance the security under the nonce-misuse scenario. Specifically, we XOR the secret key K_0 and K_1 to two branches after the initialization, e.g., we let

$$S[5] = S[5] \oplus K_0\,,$$
$$S[6] = S[6] \oplus K_1\,.$$

Remind that this countermeasure never invalidates our state-recovery attack. It makes attackers non-trivial to recover the secret key even if they recover the internal state.

Our attack reveals the whole state of one output of the initialization at the cost of 2^{128} decryption queries. The designers of Rocca did not analyze the initialization in such a setting. Therefore, whether 20 rounds are sufficient or not has to be re-evaluated to see the feasibility of the key-recovery attack under such a scenario. Although we do not claim, we expect that 20 rounds are sufficient.

We additionally recommend XORing the secret key K_0 and K_1 to two branches before the finalization, e.g., we let

$$S[1] = S[1] \oplus K_0\,,$$
$$S[2] = S[2] \oplus K_1\,.$$

This replacement makes attackers non-trivial to mount a universal forgery attack even if they recover the internal state. Note that the number of required decryption queries for our attack already exceeds the claimed 128-bit security against forgery attacks. Therefore, this tweak is not necessary to sound the security claims of Rocca. However, considering practical risks under nonce-misuse scenario, we recommend that the finalization also involves the secret key.

We stress that this countermeasure is for the 256-bit security against our key-recovery attack. This countermeasure is not helpful to achieve 256-bit security against distinguishing attacks in Claim 1. Besides, note that message-recovery attacks or universal forgery attacks are still possible at the cost of about 2^{128} even if this countermeasure is adopted, since the internal state can be recovered.

The indistinguishability security is strong and demanding, and we focus on the theoretical (in)feasibility of such strong security in the next section.

5 (In)feasibility of Achieving the Security Claim

In this section, we consider the feasibility and infeasibility of achieving Claim 1 in the provable security paradigm. We distinguish IND-CPA and IND-CCA for the security notion of distinguishing attacks, and we consider INT-CTXT as the security notion of forgery attacks.

We first define the relevant notions following [Rog02, Rog04b, BN08, ABL+14]. Let $\mathcal{A}$ be an adversary[6] and $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ be an AEAD scheme. The encryption algorithm Enc_K takes a tuple (K, N, AD, M) of a key, nonce, AD, and a message as input, and returns $(C, T) = \mathsf{Enc}_K(N, AD, M)$, a ciphertext and a tag. The decryption algorithm Dec_K takes (K, N, AD, C, T) as input, and returns $M = \mathsf{Dec}_K(N, AD, C, T)$ or $\perp$, where the latter indicates rejection. We consider the following advantage functions on an adversary $\mathcal{A}$:

$$\begin{cases} \mathbf{Adv}^{\text{ind-cpa}}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathsf{Enc}_K,\perp} \Rightarrow 1] - \Pr[\mathcal{A}^{\$,\perp} \Rightarrow 1] \\ \mathbf{Adv}^{\text{ind-cca}}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathsf{Enc}_K,\mathsf{Dec}_K} \Rightarrow 1] - \Pr[\mathcal{A}^{\$,\mathsf{Dec}_K} \Rightarrow 1] \\ \mathbf{Adv}^{\text{int-ctxt}}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathsf{Enc}_K,\mathsf{Dec}_K} \Rightarrow 1] - \Pr[\mathcal{A}^{\mathsf{Enc}_K,\perp} \Rightarrow 1] \\ \mathbf{Adv}^{\text{aead}}_{\Pi}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathsf{Enc}_K,\mathsf{Dec}_K} \Rightarrow 1] - \Pr[\mathcal{A}^{\$,\perp} \Rightarrow 1] \end{cases}$$

Here, \$ denotes an oracle that returns a uniform random bits with the same length as Enc_K, and $\perp$ denotes an oracle that always returns the reject symbol $\perp$. We only consider nonce-respecting adversaries. Following [ABL+14], we assume that if $\mathcal{A}$ receives (C, T) for an encryption query (N, AD, M), then $\mathcal{A}$ does not make a decryption query (N, AD, C, T), and in the IND-CCA notion, if $\mathcal{A}$ receives $M \neq \perp$ for a decryption query (N, AD, C, T), then $\mathcal{A}$ does not make an encryption query (N, AD, M).

[6] Throughout this section, we use "adversary" for an attacker.

Note the implications

$$\mathbf{Adv}^{\text{ind-cca}}_{\Pi}(\mathcal{A}) \le \mathbf{Adv}^{\text{ind-cpa}}_{\Pi}(\mathcal{A}) + 2\mathbf{Adv}^{\text{int-ctxt}}_{\Pi}(\mathcal{A}),$$
$$\mathbf{Adv}^{\text{aead}}_{\Pi}(\mathcal{A}) \le \mathbf{Adv}^{\text{ind-cpa}}_{\Pi}(\mathcal{A}) + \mathbf{Adv}^{\text{int-ctxt}}_{\Pi}(\mathcal{A}),$$

showing the soundness of considering IND-CPA and INT-CTXT, which are the basis of wide adoption of this combination. See [BN08] for the first one. However, as discussed in Sect. 1, when interpreted as the bit security, they only ensure $\min\{k_1, k_2\}$-bit IND-CCA or AEAD security when an AEAD scheme has k_1-bit IND-CPA and k_2-bit INT-CTXT security.

Recall that we denote the concatenation of bit strings X and Y by $X \,\|\, Y$, and $|X|$ denotes the bit length of X.

Feasibility of 256-bit IND-CPA and 128-bit INT-CTXT Security. We first point out that achieving 256-bit IND-CPA security and 128-bit INT-CTXT security is possible with known approaches, where the tag length is 128 bits.

Proposition 1. *There exists an AEAD scheme* Π *with* 256*-bit IND-CPA security and* 128*-bit INT-CTXT security.*

This proposition can be confirmed by various known AEAD modes of operations that use a $4n$-bit block cipher or a $5n$-bit cryptographic permutation (with an n-bit rate) and ensure up-to-birthday-bound security, when they have n-bit tags. For example, we can extend GCM so that it works with a $4n = 512$-bit block cipher together with an extended GHASH defined by multiplications over $\mathbb{F}_{2^{4n}}$. We use (say) $2n$-bit nonces and a $2n$-bit block counter where their concatenation forms the initial counter-mode block input, just as in the original GCM using 96-bit nonces and a 32-bit block counter. The tag length is $\tau = 128$ bits. Such an extended version of GCM has the following security bounds:

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{GCM}[E]}(\mathcal{A}_{\text{priv}}) \le \mathbf{Adv}^{\text{prp}}_{E}(\mathcal{A}') + \frac{0.5(\sigma_e + q_e + 1)^2}{2^{4n}},$$
$$\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{GCM}[E]}(\mathcal{A}_{\text{auth}}) \le \mathbf{Adv}^{\text{prp}}_{E}(\mathcal{A}'') + \frac{0.5(\sigma_e + q_e + 1)^2}{2^{4n}} + \frac{q_d(\ell_A + 1)}{2^{\tau}},$$

where $\mathcal{A}_{\text{priv}}$ denotes the privacy adversary using σ_e total encrypted blocks and q_e encryption queries, and $\mathcal{A}_{\text{auth}}$ denotes the authenticity adversary using σ_e total encrypted and decrypted blocks and q_e encryption and q_d decryption queries, with the maximum of ℓ_A blocks of AD in any query, for $4n$-bit blocks. The first terms of the bounds show the indistinguishability of the underlying block cipher E from a $4n$-bit random permutation (i.e., pseudorandom permutation advantage) and are assumed to be negligible. The proofs are mostly immediate from the original proofs [IOM12, Corollaries 3 and 4].

Similar results can be obtained for OCB [KR11], if the underlying (small) constant multiplications over $\mathbb{F}_{2^{4n}}$ (so-called doubling and tripling etc.) fulfill certain distinctness conditions[7]. Rogaway [Rog04a] presented concrete instances for the case of $\mathbb{F}_{2^{64}}$ and $\mathbb{F}_{2^{128}}$, and Granger et al. [GJMN16] extended the analysis to $\mathbb{F}_{2^{512}}$ to $\mathbb{F}_{2^{1024}}$, which can be useful for our case. In fact, [GJMN16] defined OPP, which is a permutation-based variant of (large-block) OCB. It gives an AEAD scheme based on a $4n = 512$-bit permutation having 256-bit IND-CPA and 128-bit INT-CTXT security if tags are truncated to $\tau = 128$ bits. To confirm this claim, we refer to the bound shown in the full version of the paper [GJMN15, Theorem 5]. It fixed a simple error in the proof of the original paper and presented a security bound for the unified AEAD security notion ($\mathbf{Adv}^{\text{aead}}_{\mathsf{OPP}}(\mathcal{A})$) for OPP. When a 512-bit permutation is used and the key length is $k = 256$, one can confirm the above

[7] We also need to consider the extended form of the component called stretch-then-shift universal hash function that processes a nonce.

256-bit IND-CPA claim by making the number of decryption queries being zero in the unified bound. The 128-bit INT-CTXT bound is clear since the unified bound contains $q_d/2^\tau$ and a unified bound is essentially the sum of IND-CPA and INT-CTXT bounds.

Sponge-based AEAD, such as duplex sponge [BDPA11], would also allow similar security bounds by using a permutation of an appropriate size. That is, a permutation of size $5n$ bits with an n-bit rate and a tag length of n bits is enough to achieve the bound.

Therefore, there are schemes with 256-bit IND-CPA security and 128-bit INT-CTXT security. However, as discussed in Sect. 1, the high indistinguishably security is guaranteed only when the decryption oracle is not available to the adversary. If the adversary has the decryption oracle, the security guarantee (in indistinguishability or in the unified notion) degrades to 128 bits. In general, this does not necessarily imply the existence of an attack with 2^{128} complexity. However, we next show that distinguishing attacks are possible in the IND-CCA security notion.

Infeasibility of 256-bit IND-CCA and 128-bit INT-CTXT Security. We consider a class of AEAD schemes called an online AEAD scheme [BBKN12, ABL+14]. Intuitively, in these schemes, the i-th output block depends only on the first i blocks of the input. This class includes all the schemes stated above, Rocca, and its variant in Sect. 4.2, and we show that they cannot achieve 256-bit IND-CCA security.

Let m be a positive integer. Following [BBKN12, ABL+14], we call an AEAD scheme $\Pi = (\mathsf{Enc}, \mathsf{Dec})$ *m-online* if it satisfies the following condition: For any tuple (K, N, AD), there exist functions $f_1, f_2, \ldots$ of which codomain is $\mathbb{F}_2^m$ and another function g such that

$$\mathsf{Enc}_K(N, AD, M) = f_1(M_1) \,\|\, f_2(M_1, M_2) \,\|\, \cdots \,\|\, f_\ell(M_1, \ldots, M_\ell) \,\|\, g(M) \tag{2}$$

holds for any M, where $M = M_1 \,\|\, \cdots \,\|\, M_\ell \,\|\, M'$, $M_1, \ldots, M_\ell \in \mathbb{F}_2^m$, and $M' \in \mathbb{F}_2^{m'}$ $(0 \le m' < m)$.

The following proposition roughly shows that a τ-bit tag online AEAD scheme cannot achieve more than τ-bit IND-CCA security. In particular, 128-bit tag AEAD schemes based on the sponge(-like) construction, including Rocca (and its variant mentioned in Sect. 4.2), cannot achieve 256-bit IND-CCA security.

Proposition 2. *Let Π be an m-online AEAD satisfying $|\mathsf{Enc}_K(N, AD, M)| = |M| + \tau$ for a fixed constant $\tau > 0$. Let Enc'_K be a truncated version of Enc_K discarding outputs of g (see Eq. (2)), and assume the restriction of $\mathsf{Enc}'_K(N, AD, \cdot)$ to $\mathbb{F}_2^{2m}$ is a permutation for any choice of N and AD. Then there exists an adversary $\mathcal{A}$ making at most 2^τ decryption queries and a single encryption query such that $\mathbf{Adv}_\Pi^{\text{ind-cca}}(\mathcal{A}) = 1 - 1/2^m$ holds.*

Proof. In what follows we assume no AD is involved in encryption queries nor decryption queries.

Let $\mathcal{A}$ be an adversary running as follows: (1) Choose a nonce N and $C_0, C_1 \in \mathbb{F}_2^m$ arbitrarily. (2) For each $T \in \mathbb{F}_2^\tau$, query $(N, C_0 \,\|\, C_1 \,\|\, T)$ to the decryption oracle. If a $2m$-bit message $M = M_0 \,\|\, M_1 \neq \perp$ is returned $(M_0, M_1 \in \mathbb{F}_2^m)$, proceed to the next step. (3) Take an m-bit string $M_1' \neq M_1$ and query $(N, M_0 \,\|\, M_1')$ to an encryption oracle. If the first block (the first m bits) of the response matches C_0, output 1. Otherwise output 0.

Suppose we run the real or ideal world for the definition of IND-CCA security of Π with the above $\mathcal{A}$. Since the restriction of $\mathsf{Enc}'_K(N, \cdot)$ to $\mathbb{F}_2^{2m}$ is a permutation by assumption, for arbitrary choice of N, C_0, and C_1 in the first step of $\mathcal{A}$, there exists a unique $M \in \mathbb{F}_2^{2m}$ such that $\mathsf{Enc}'_K(N, M) = C_0 \,\|\, C_1$. This implies the existence of $T \in \mathbb{F}_2^\tau$ satisfying $\mathsf{Enc}_K(N, M) = C_0 \,\|\, C_1 \,\|\, T$. In particular, $\mathcal{A}$ receives M at some point of the second step and proceed to the third step. These arguments hold in both of the real and ideal worlds.

Let C_0' be the first block of the response that $\mathcal{A}$ receives from an encryption oracle in its third step. In the real world, $C_0' = C_0$ always holds since Π is m-online and $\mathcal{A}$ outputs 1

with probability 1. However, in the ideal world, we have $\Pr[C'_0 = C_0] = 2^{-m}$ because the ideal encryption oracle returns a random output. Hence $\mathbf{Adv}^{\text{ind-cca}}_{\Pi}(\mathcal{A}) = 1 - 1/2^m$. □

Proposition 2 shows that distinguishing attacks are possible against Rocca and its variant in Sect. 4.2, while for these schemes, message-recovery attacks are also possible as the internal state can be recovered.

Now Proposition 2 rules out efficient solutions, and we next consider an offline construction to see the feasibility.

Feasibility of 256-bit IND-CCA and 128-bit INT-CTXT Security. We show that, with the Encode-then-Encipher approach [BR00], we can simultaneously achieve 256-bit IND-CCA security and 128-bit INT-CTXT security.

Before presenting our theorem and the proof, let us fix some notations. Let n be a positive integer and X be an n-bit string. For a positive integer $x \leq n$, $\mathtt{msb}_x(X)$ (resp. $\mathtt{lsb}_x(X)$) denotes the truncation of X to its x most (resp. least) significant bits. A tweakable block cipher (TBC) [LRW11] is a keyed function $\widetilde{E} : \mathcal{K} \times \mathcal{TW} \times \mathcal{M} \to \mathcal{M}$, where $\mathcal{K}$ is a key space, $\mathcal{TW}$ is a tweak space, and $\mathcal{M}$ is a message space. A tweak is a public parameter and $\widetilde{E}(K, TW, \cdot)$ is a permutation on $\mathcal{M}$ for $\forall (K, TW) \in \mathcal{K} \times \mathcal{TW}$. We also write $\widetilde{E}_K$ to mean $\widetilde{E}(K, \cdot, \cdot)$. Let $\mathsf{Perm}(n)$ denote the set of all permutations on $\mathbb{F}_2^n$. An n-bit tweakable permutation with a tw-bit tweak is a function $\pi : \mathbb{F}_2^{tw} \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ such that $\pi(TW, \cdot) \in \mathsf{Perm}(n)$ for $\forall TW \in \mathbb{F}_2^{tw}$. The set of all n-bit tweakable permutations with a tw-bit tweak is denoted by $\mathsf{TPerm}(tw, n)$. Let $\widetilde{\mathsf{P}}$ such that $\widetilde{\mathsf{P}} \xleftarrow{\$} \mathsf{TPerm}(tw, n)$ be a tweakable uniform random permutation (TURP). The (strong) security of a TBC $\widetilde{E}$ is defined as the advantage function $\mathbf{Adv}^{\text{tsprp}}_{\widetilde{E}}(\mathcal{A}) = \Pr[\mathcal{A}^{\widetilde{E}_K, \widetilde{E}_K^{-1}} \Rightarrow 1] - \Pr[\mathcal{A}^{\widetilde{\mathsf{P}}, \widetilde{\mathsf{P}}^{-1}} \Rightarrow 1]$.

The following theorem shows that achieving 256-bit IND-CCA security and 128-bit INT-CTXT security is possible.

Theorem 1. *Let $\mathcal{A}$ be an adversary that makes q_e encryption queries and q_d decryption queries. There exists an AEAD scheme Π such that $\mathbf{Adv}^{\text{ind-cca}}_{\Pi}(\mathcal{A}) \leq q_d/2^{256}$ and $\mathbf{Adv}^{\text{int-ctxt}}_{\Pi}(\mathcal{A}) \leq q_d/2^{128}$, where $q_e \leq 2^{256}$ in both bounds, $q_d \leq 2^{255}$ in the IND-CCA bound, and $q_d \leq 2^{127}$ in the INT-CTXT bound.*

Proof. Let $\widetilde{E}_K(\cdot, \cdot)$ be a TBC with a 256-bit input/output and a 256-bit tweak. We prove the Encode-then-Encipher (EtE) scheme based on $\widetilde{E}$ fulfills the bounds in Theorem 1. For the sake of ease, we assume that EtE only accepts a fixed message length of $m := 128$ bits and there is no AD. We later cover the general case. For a 256-bit nonce N, a 128-bit message M, and a 256-bit ciphertext C, we define $\mathsf{EtE.Enc}_K(N, M) := \widetilde{E}_K(N, 0^{128} \| M)$, and $\mathsf{EtE.Dec}_K(N, C) = \mathtt{lsb}_m(\widetilde{E}_K^{-1}(N, C))$ if $\mathtt{msb}_{128}(\widetilde{E}_K^{-1}(N, C)) = 0^{128}$ holds, otherwise, $\mathsf{EtE.Dec}_K(N, C) = \perp$.

For IND-CCA security, we firstly convert $\widetilde{E}$ to the TURP $\widetilde{\mathsf{P}}$ having the same tweak and message spaces. This step adds $\mathbf{Adv}^{\text{tsprp}}_{\widetilde{E}}(\mathcal{A}')$ to the bound, which we assume to be small. Let $(N_1, M_1, C_1), \ldots, (N_{q_e}, M_{q_e}, C_{q_e})$ be the sequence of encryption queries, and $(N'_1, C'_1, M'_1), \ldots, (N'_{q_d}, C'_{q_d}, M'_{q_d})$ be the sequence of decryption queries. Note that all N_i for $1 \leq i \leq q_e$ are distinct since the adversary follows the nonce-respecting setting, and M'_i for $1 \leq i \leq q_d$ could be $\perp$. The adversary can distinguish the ideal world from the real world if and only if any one of the following cases occurs:

Case-1 The adversary obtains (N', C', M') from the decryption oracle and subsequently queries (N', M) such that $M \neq M'$ to the encryption oracle and obtains ciphertext C such that $C = C'$.

Case-2 The adversary obtains (N, M, C) from the encryption oracle and subsequently queries (N, C') such that $C' \neq C$ to the decryption oracle and obtains message M' such that $M = M'$.

There is no bad event between encryption queries because the adversary follows the nonce-respecting setting and $\widetilde{\mathsf{P}}$ takes a nonce as tweak input. Also, there is no bad event between decryption queries because both worlds have the same (real decryption) scheme. In the real world, Case-1 and Case-2 do not appear since $\widetilde{\mathsf{P}}$ is a random permutation under the fixed nonce. We evaluate $\Pr[\mathsf{Case\text{-}1} \cup \mathsf{Case\text{-}2}]$ in the ideal world.

Let q_i denote the number of decryption queries whose nonce equals the nonce of the i-th encryption query N_i, thus $\sum_{i=1}^{q_e} q_i \le q_d$. We then split q_i into two variables, $q_{i,\mathsf{b}}$ and $q_{i,\mathsf{a}}$. The former is the number of the decryption queries with the nonce N_i before the adversary obtains (N_i, M_i, C_i), and the latter is the one after that, thus $q_i = q_{i,\mathsf{b}} + q_{i,\mathsf{a}}$. For the i-th encryption query, the probability of Case-1 is $q_{i,\mathsf{b}}/2^{256}$. Note that M' in the decryption queries can be $\perp$, i.e., there is no need for the adversary to first succeed in forgery. For Case-2, we renumber the decryption query, whose nonce is the same as N_i, as $(N_i, C_1'^{i,\mathsf{b}}, M_1'^{i,\mathsf{b}}), \ldots, (N_i, C_{q_{i,\mathsf{b}}}'^{i,\mathsf{b}}, M_{q_{i,\mathsf{b}}}'^{i,\mathsf{b}})$ and $(N_i, C_1'^{i,\mathsf{a}}, M_1'^{i,\mathsf{a}}), \ldots, (N_i, C_{q_{i,\mathsf{a}}}'^{i,\mathsf{a}}, M_{q_{i,\mathsf{a}}}'^{i,\mathsf{a}})$. For $j \in \{1, \ldots, q_{i,\mathsf{a}}\}$, suppose that $M_1'^{i,\mathsf{a}} \neq M_i, \ldots, M_{j-1}'^{i,\mathsf{a}} \neq M_i$. At the decryption query $(N_i, C_j'^{i,\mathsf{a}}, M_j'^{i,\mathsf{a}})$, the probability of $M_j'^{i,\mathsf{a}} = M_i$ is at most $1/(2^{256} - q_{i,\mathsf{b}} - j)$ since the adversary has to query 256-bit ciphertext $C_j'^{i,\mathsf{a}}$, such that $\widetilde{\mathsf{P}}^{-1}(N_i, C_j'^{i,\mathsf{a}}) = 0^{128} \,\|\, M_i$ and $C_j'^{i,\mathsf{a}} \notin \{C_i, C_1'^{i,\mathsf{b}}, \ldots, C_{q_{i,\mathsf{b}}}'^{i,\mathsf{b}}, C_1'^{i,\mathsf{a}}, \ldots, C_{j-1}'^{i,\mathsf{a}}\}$, to the decryption oracle. Thus, assuming $q_{i,\mathsf{b}} + q_{i,\mathsf{a}} \le q_d \le 2^{255}$, the probability of Case-2 is at most $\sum_{i=1}^{q_e} \sum_{j=1}^{q_{i,\mathsf{a}}} 1/(2^{256} - q_{i,\mathsf{b}} - j) \le \sum_{i=1}^{q_e} 2q_{i,\mathsf{a}}/2^{256}$. Therefore, $\Pr[\mathsf{Case\text{-}1} \cup \mathsf{Case\text{-}2}] \le \sum_{i=1}^{q_e} q_{i,\mathsf{b}}/2^{256} + \sum_{i=1}^{q_e} 2q_{i,\mathsf{a}}/2^{256} = \sum_{i=1}^{q_e}(q_i/2^{256} + q_{i,\mathsf{a}}/2^{256}) \le 2q_d/2^{256}$.

For INT-CTXT security, we use $\widetilde{\mathsf{P}}$ for $\widetilde{E}$. This adds a term $\mathbf{Adv}^{\mathrm{tsprp}}_{\widetilde{E}}(\mathcal{A}'')$, which is assumed to be small. Now it is easy to see $\mathbf{Adv}^{\text{int-ctxt}}_{\mathsf{EtE}}(\mathcal{A}) \le \sum_{j=1}^{q_d} 2^{128}/(2^{256} - j) \le 2q_d/2^{128}$, by following a similar argument to Case-2 of the IND-CCA analysis. Note that the constant 2 in the IND-CCA and INT-CTXT bounds are reduced to 1 by increasing m by 1, and assuming a perfectly secure TBC, we obtain the bounds of the theorem.

For the general case, where there is variable-length associated data AD and a variable-length message M, the proof is easily extended. We assume that $\widetilde{E}$ accepts a variable-length tweak and a variable-length message block, and use $(N \,\|\, AD)$ as a tweak and use $(0^{128} \,\|\, \mathsf{pad}(M))$ as an input block with a certain injective padding pad that ensures $|\mathsf{pad}(M)| > 128$ for any M. □

We do not specify the choice of $\widetilde{E}$, which could be built from a scratch or could be a mode of operation such as [HR04, WFW05] (instantiated with a block cipher with an appropriate block length). If $\widetilde{E}$ has 256-bit security against key-recovery attacks, we see that the claim against key-recovery attacks in Claim 1 can also be achieved.

Finally, we remark that Theorem 1 can be extended to handle a general case. For any positive integer k_1, k_2 such that $k_1 > k_2$, we can simultaneously achieve k_1-bit IND-CCA security and k_2-bit INT-CTXT security. Let $n \coloneqq k_1$, $\tau \coloneqq k_2$, $m \coloneqq n - \tau$. Let $\widetilde{E}$ be a TBC with an n-bit input/output and an n-bit tweak. We define a generalized EtE, which we call gEtE, as $\mathsf{gEtE.Enc}_K(N, M) \coloneqq \widetilde{E}_K(N, 0^\tau \,\|\, M)$, and $\mathsf{gEtE.Dec}_K(N, C) = \mathtt{lsb}_m(\widetilde{E}_K^{-1}(N, C))$ if $\mathtt{msb}_\tau(\widetilde{E}_K^{-1}(N, C)) = 0^\tau$ holds, otherwise, $\mathsf{EtE.Dec}_K(N, C) = \perp$. We can prove that gEtE has k_1-bit IND-CCA security and k_2-bit INT-CTXT in the same manner as the proof of Theorem 1.

6 Conclusions

In this paper, we first presented a key-recovery attack on Rocca, where the time complexity is about 2^{128} and the success probability is almost 1. This shows that, in terms of key-recovery, Rocca has 128-bit security. We then considered extensions of the attack to various security models and discussed countermeasures. We also studied a theoretical question of

achieving the security claim of Rocca, and showed that the Encode-then-Encipher approach gives a feasible result at the cost of efficiency.

In Rocca, only one nonce-repeated pair is enough to recover the internal state and the secret key with the practical time complexity, which is the main reason of the success of the attack. Even if the nonce-misuse security is optional, this casts the importance of maintaining a level of security under the nonce-misuse scenario. Involving the secret key in the initialization and finalization is an effective way, as the security against key-recovery attacks improves and the cost is negligible, while message-recovery attacks or universal forgery attacks are still possible at the cost of about 2^{128} complexity. Related to this, Rocca defines a raw encryption mode, which is obtained by removing the process of AD and the finalization, and our attack implies that if an attacker has a decryption oracle for this raw encryption mode, this immediately allows a key-recovery attack with 2 decryption queries.

As we discussed in Sect. 3.1, having a strong privacy security bound and a weaker authenticity bound is relevant in practical contexts. For instance, given some AEAD scheme, possibly used in IoT applications, one may want to truncate the tag to reduce the bandwidth to save energy for data transmission, hoping that the impact on the privacy security bound is limited. Rocca is not suitable for this purpose, as its tag length directly degrades the security. The Encode-then-Encipher scheme has an efficiency issue, and designing an efficient scheme retaining a level of CCA security even with tag truncation is an interesting question.

Acknowledgments

The authors would like to thank the designers of Rocca for feedback on the early draft of the paper. Akinori Hosoyamada, Tetsu Iwata, Kazuhiko Mimematsu, and Yosuke Todo would like to thank the organizers of Dagstuhl Seminar 22141, Symmetric Cryptography, for inviting them to the seminar. Although they had to cancel the attendance and none of them could attend the seminar due to the COVID-19 pandemic, the invitation initiated the collaboration among them, leading to the results of this paper.

References

[ABL+14] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 105–125. Springer, 2014.

[BBKN12] Mihir Bellare, Alexandra Boldyreva, Lars R. Knudsen, and Chanathip Namprempre. On-line ciphers and the hash-cbc constructions. *J. Cryptol.*, 25(4):640–679, 2012.

[BDD+12] Charles Bouillaguet, Patrick Derbez, Orr Dunkelman, Pierre-Alain Fouque, Nathan Keller, and Vincent Rijmen. Low-data complexity attacks on AES. *IEEE Trans. Inf. Theory*, 58(11):7002–7017, 2012.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011,*

Revised Selected Papers, volume 7118 of *Lecture Notes in Computer Science*, pages 320–337. Springer, 2011.

[BN08] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptol.*, 21(4):469–491, 2008.

[BR00] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.

[EJMY19] Patrik Ekdahl, Thomas Johansson, Alexander Maximov, and Jing Yang. A new SNOW stream cipher called SNOW-V. *IACR Trans. Symmetric Cryptol.*, 2019(3):1–42, 2019.

[FFL12] Ewan Fleischmann, Christian Forler, and Stefan Lucks. Mcoe: A family of almost foolproof on-line authenticated encryption schemes. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 196–215. Springer, 2012.

[GJMN15] Robert Granger, Philipp Jovanovic, Bart Mennink, and Samuel Neves. Improved masking for tweakable blockciphers with applications to authenticated encryption. *IACR Cryptol. ePrint Arch.*, page 999, 2015.

[GJMN16] Robert Granger, Philipp Jovanovic, Bart Mennink, and Samuel Neves. Improved masking for tweakable blockciphers with applications to authenticated encryption. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 263–293. Springer, 2016.

[GRV21] Emiljano Gjiriti, Reza Reyhanitabar, and Damian Vizár. Power yoga: Variable-stretch security of CCM for energy-efficient lightweight iot. *IACR Trans. Symmetric Cryptol.*, 2021(2):446–468, 2021.

[HR04] Shai Halevi and Phillip Rogaway. A parallelizable enciphering mode. In Tatsuaki Okamoto, editor, *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, volume 2964 of *Lecture Notes in Computer Science*, pages 292–304. Springer, 2004.

[HRRV15] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizár. Online authenticated-encryption and its nonce-reuse misuse-resistance. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 493–517. Springer, 2015.

[IOM12] Tetsu Iwata, Keisuke Ohashi, and Kazuhiko Minematsu. Breaking and repairing GCM security proofs. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 31–49. Springer, 2012.

[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.

[Kha22] Mustafa Khairallah. Security of COFB against chosen ciphertext attacks. *IACR Trans. Symmetric Cryptol.*, 2022(1):138–157, 2022.

[KR11] Ted Krovetz and Phillip Rogaway. The software performance of authenticated-encryption modes. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, volume 6733 of *Lecture Notes in Computer Science*, pages 306–327. Springer, 2011.

[LRW11] Moses D. Liskov, Ronald L. Rivest, and David A. Wagner. Tweakable block ciphers. *J. Cryptol.*, 24(3):588–613, 2011.

[Mèg19] Alexandre Mège. [lwc-forum] tag length impact on confidentiality security. Comment to NIST mailing list, November 2019. https://groups.google.com/a/list.nist.gov/g/lwc-forum/c/2aOH-HQHgqU/m/EtjdRFSmBQAJ.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[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 Nikolić. Tiaoxin-346. CAESAR submission, 2014.

[NL05] Yoav Nir and Adam Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539, May 2005.

[NRS13] Chanathip Namprempre, Phillip Rogaway, and Tom Shrimpton. AE5 security notions: Definitions implicit in the CAESAR call. *IACR Cryptol. ePrint Arch.*, page 242, 2013.

[NRS14] Chanathip Namprempre, Phillip Rogaway, and Thomas Shrimpton. Reconsidering generic composition. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 257–274. Springer, 2014.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 98–107. ACM, 2002.

[Rog04a] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, volume 3329 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[Rog04b] Phillip Rogaway. Nonce-based symmetric encryption. In Bimal K. Roy and Willi Meier, editors, *Fast Software Encryption, 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004, Revised Papers*, volume 3017 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2004.

[RS06] Phillip Rogaway and Thomas Shrimpton. A provable-security treatment of the key-wrap problem. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2006.

[RVV16] Reza Reyhanitabar, Serge Vaudenay, and Damian Vizár. Authenticated encryption with variable stretch. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 396–425, 2016.

[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+22] 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.*, page 116, 2022.

[WFW05] Peng Wang, Dengguo Feng, and Wenling Wu. HCTR: A variable-input-length enciphering mode. In Dengguo Feng, Dongdai Lin, and Moti Yung, editors, *Information Security and Cryptology, First SKLOIS Conference, CISC 2005, Beijing, China, December 15-17, 2005, Proceedings*, volume 3822 of *Lecture Notes in Computer Science*, pages 175–188. Springer, 2005.

[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: State-Recovery Attack against Rocca Using Meet-in-the-Middle Technique

We provide a test case for the state-recovery attack against Rocca using meet-in-the-middle technique. First, we generate a secret key K ($= K_0 \,\|\, K_1$), a nonce N, associated data AD, and a message M at random, and then simulate the encryption oracle to get the ciphertext C from a tuple (K, N, AD, M):

secret key K:
```
FA 1A 5A 8C 12 B6 07 FD F7 B1 16 CC 87 A1 F0 22
35 5A 90 01 57 81 93 74 19 8E 35 B7 47 C9 0C 2E
```

nonce N:
```
38 75 A1 88 7A EF EE 04 0E F5 B9 43 2E 62 8F 2E
```

associated data AD:
```
02 33 B6 B4 72 C8 1A C6 D9 1C C8 86 B1 77 E6 72
EF B9 F8 BD D4 F2 98 93 28 05 6F 91 B2 F2 CF 1A
51 73 D4 C6 55 FD 98 A9 41 05 16 2C 73 44 5B 6E
66 F9 99 49 EE CE 66 1D F4 25 F9 83 B2 78 49 A6
```

message M:
```
C9 53 0D 7D 35 6C F7 52 E8 8E F6 F4 95 27 0F 62
1B 1B B5 C8 ED D7 E0 C7 9C 13 B2 02 23 4C 87 3D
09 21 C9 52 F8 FC E0 F3 18 58 FB 59 59 A3 91 0E
25 7F D2 D1 D8 AE B9 09 28 DF 2C B2 D6 5F 2C 2F
2A C4 A7 CB 7A 1C 53 9E 58 AB 5D 15 96 05 EE F7
C4 F0 08 F9 66 05 66 18 54 C8 3C FC 74 7F 90 5A
0C 66 8B B7 32 75 AA D8 5F 97 16 2C 5D 77 03 42
5E 71 DC 74 FA 7D 21 BA A4 D2 FA 5F 9B A9 B4 ED
```

ciphertext C:
```
7A 70 F8 0A 10 D1 86 7B 0E 6B 5C 53 8D 87 27 BC
1B A7 16 DA 26 F5 B8 BB 70 D9 27 A9 77 9A F0 4D
DD A6 9B 2C DC 2E E6 AE 6B 28 A8 25 61 D7 EE 26
04 13 EC C6 36 55 82 A3 B2 40 30 84 FE 96 C2 37
45 F8 B8 A3 27 E5 0A 96 9A EB E8 74 E2 C2 7C 1E
3D 6D 5B 99 9D CA C1 EC 12 18 37 28 3C 4F 03 64
35 BD 31 5F F2 E2 BB 32 6F 65 C2 36 30 7F F9 9A
C0 38 60 62 2E C1 64 98 32 AA FF 6B AD D0 DA 83
```

Next, we inject a proper difference Δ to the ciphertext C, and then simulate the decryption oracle to get the message M' from a tuple $(K, N, AD, C \oplus \Delta)$:

difference Δ to the ciphertext C:
```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

ciphertext $C \oplus \Delta$:

```
7A 70 F8 0A 10 D1 86 7B 0E 6B 5C 53 8D 87 27 BC
1A A6 17 DB 27 F4 B9 BA 71 D8 26 A8 76 9B F1 4C
DD A6 9B 2C DC 2E E6 AE 6B 28 A8 25 61 D7 EE 26
04 13 EC C6 36 55 82 A3 B2 40 30 84 FE 96 C2 37
45 F8 B8 A3 27 E5 0A 96 9A EB E8 74 E2 C2 7C 1E
3D 6D 5B 99 9D CA C1 EC 12 18 37 28 3C 4F 03 64
35 BD 31 5F F2 E2 BB 32 6F 65 C2 36 30 7F F9 9A
C0 38 60 62 2E C1 64 98 32 AA FF 6B AD D0 DA 83
```

message M':

```
C9 53 0D 7D 35 6C F7 52 E8 8E F6 F4 95 27 0F 62
1A 1A B4 C9 EC D6 E1 C6 9D 12 B3 03 22 4D 86 3C
09 21 C9 52 F8 FC E0 F3 18 58 FB 59 59 A3 91 0E
24 4B 1C 7D E5 6E 7A 6D 70 E7 D3 BD B8 6E E3 12
4B 27 20 A0 8B 25 07 AA 0C E8 AE 9B 1D F4 1F 8D
F5 7F 69 EB F9 49 4E CA C0 B1 DD F3 B4 20 B9 FC
B9 01 88 64 6D 9E FB 8A 49 18 C6 A4 85 B4 73 7D
34 7F FA 02 5B 9B 1C F9 C4 36 E2 48 DC 88 25 1A
```

Finally, we use the "nonce-repeated" pair (M, C) and $(M', C \oplus \Delta)$ to recover the whole internal state of S_1:

```
S_1[0]: C3 0B B2 58 FF BC CC 7F EF 45 00 B8 8C 59 AF 88
S'_1[0]: C3 0B B2 58 FF BC CC 7F EF 45 00 B8 8C 59 AF 88

S_1[1]: EA DC C6 2C 25 D6 67 C9 09 E9 C7 5A F8 C1 D4 EE
S'_1[1]: EA DC C6 2C 25 D6 67 C9 09 E9 C7 5A F8 C1 D4 EE

S_1[2]: 95 5C CE 00 07 83 FE 5A 3E 6D 2C 3B 5F B6 E3 B5
S'_1[2]: 95 5C CE 00 07 83 FE 5A 3E 6D 2C 3B 5F B6 E3 B5

S_1[3]: E8 F7 D1 98 DD 92 70 15 A1 89 9F BA 9A 85 D7 E8
S'_1[3]: E8 F7 D1 98 DD 92 70 15 A1 89 9F BA 9A 85 D7 E8

S_1[4]: D7 D3 13 65 DD 64 6E B1 3F 10 C1 59 12 43 2C B2
S'_1[4]: D6 D2 12 64 DC 65 6F B0 3E 11 C0 58 13 42 2D B3

S_1[5]: 2E 8E CC 8C 41 78 24 A8 90 9B 25 12 10 08 70 8F
S'_1[5]: 2E 8E CC 8C 41 78 24 A8 90 9B 25 12 10 08 70 8F

S_1[6]: EB 40 C0 B4 72 1E D7 34 3C 44 43 0F FE EA FA E4
S'_1[6]: EB 40 C0 B4 72 1E D7 34 3C 44 43 0F FE EA FA E4

S_1[7]: B4 9C 7C B8 8D DA 4F 9E AF 31 B5 12 15 FD 82 B7
S'_1[7]: B4 9C 7C B8 8D DA 4F 9E AF 31 B5 12 15 FD 82 B7
```

In this case, we can recover the whole internal state of S_1 with a time complexity of 2^{20}. That is, all of our guesses can be narrowed down to two candidates.

B Forgery Attack against Rocca Using Longer Tag Length

Increasing the tag length is one of the most simple countermeasures against our attack. However, the designers of Rocca do not guarantee strong security that is implied by the

increased tag length against forgery attacks. Therefore, we evaluated differential trails that are available in forgery attacks: differential trail whose input and output differences are zero but a non-zero message difference is absorbed. Figures 5 and 6 show such a differential trail. There are 25 active S-boxes and all of them transit with the probability of 2^{-6}. Therefore, the probability is $2^{-6\times 25} = 2^{-150}$.

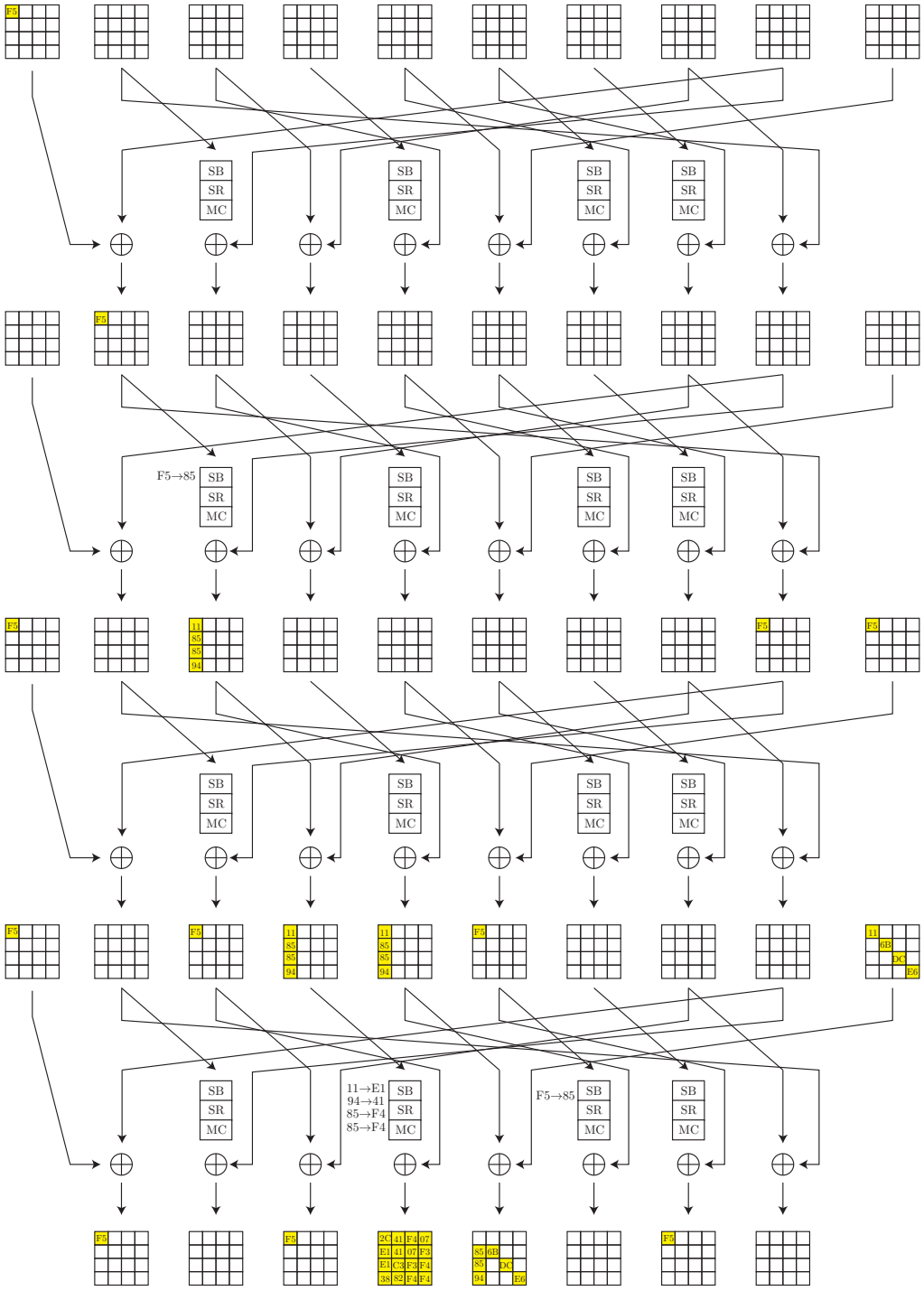

Figure 5: Top 4 rounds in the differential trail available for the forgery attack.

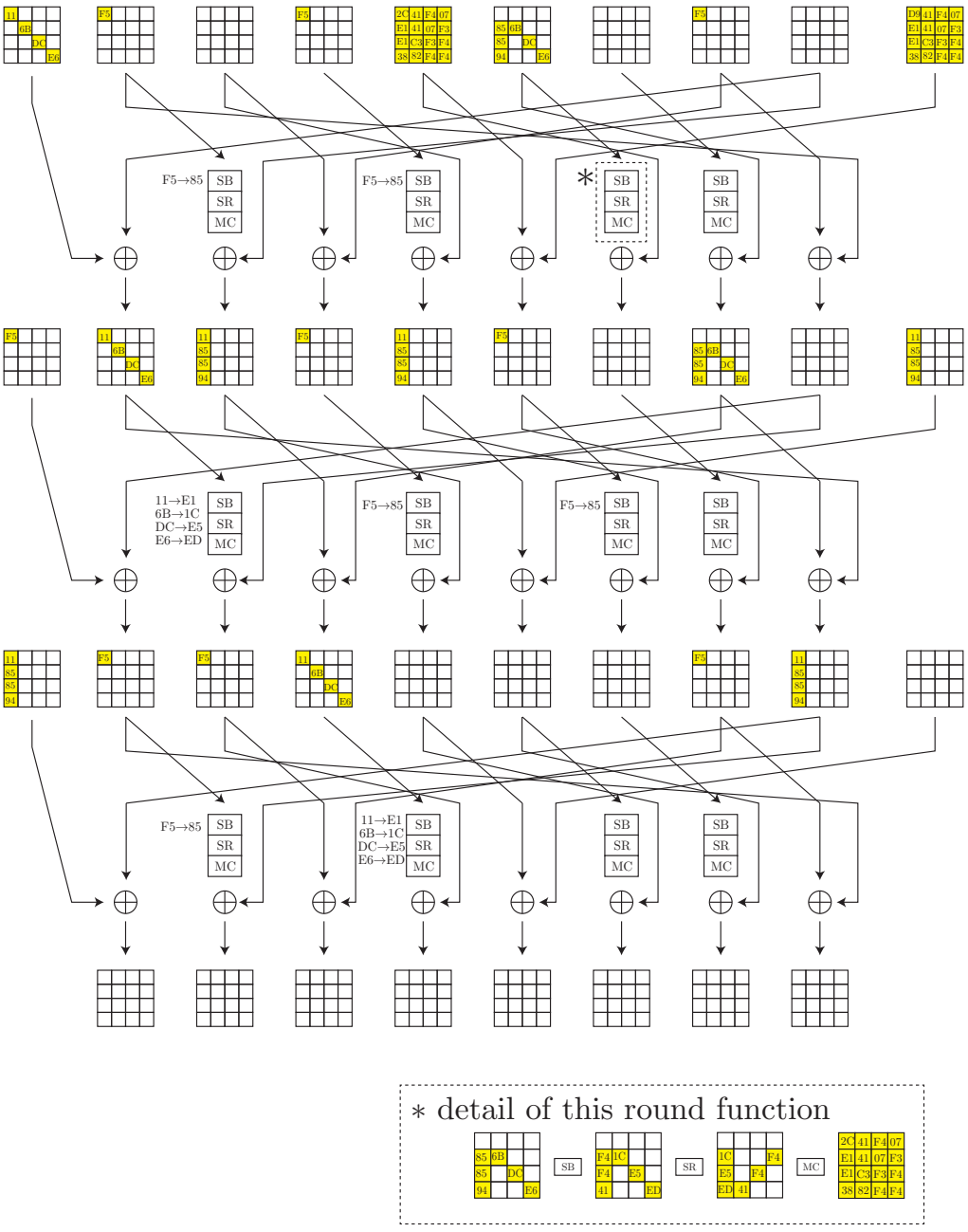

Figure 6: Bottom 3 rounds in the differential trail available for the forgery attack.