IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2024, No. 1, pp. 135–157. DOI:10.46586/tosc.v2024.i1.135-157

Key Committing Attacks against AES-based AEAD Schemes

Patrick Derbez[1], Pierre-Alain Fouque[1], Takanori Isobe[2], Mostafizar Rahman[2] and André Schrottenloher[1]

[1] Univ Rennes, Inria, Centre National de la Recherche Scientifique (CNRS), Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Rennes, France
patrick.derbez@irisa.fr, pierre-alain.fouque@irisa.fr, andre.schrottenloher@inria.fr
[2] University of Hyogo, Kobe, Japan
takanori.isobe@ai.u-hyogo.ac.jp, mrahman454@gmail.com

Abstract. Recently, there has been a surge of interest in the security of authenticated encryption with associated data (AEAD) within the context of key commitment frameworks. Security within this framework ensures that a ciphertext chosen by an adversary does not decrypt to two different sets of key, nonce, and associated data. Despite this increasing interest, the security of several widely deployed AEAD schemes has not been thoroughly examined within this framework. In this work, we assess the key committing security of several AEAD schemes. First, the AEGIS family, which emerged as a winner in the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR), and has been proposed to standardization at the IETF. A now outdated version of the draft standard suggested that AEGIS could qualify as a fully committing AEAD scheme; we prove that it is not the case by proposing a novel attack applicable to all variants, which has been experimentally verified. We also exhibit a key committing attack on Rocca-S. Our attacks are executed within the FROB game setting, which is known to be one of the most stringent key committing frameworks. This implies that they remain valid in other, more relaxed frameworks, such as CMT-1, CMT-4, and so forth. Finally, we show that applying the same attack techniques to Rocca and Tiaoxin-346 does not compromise their key-committing security. This observation provides valuable insights into the design of such secure round update functions for AES-based AEAD schemes.

Keywords: AEGIS · Key Commitment · Rocca-S · Rocca · Tiaoxin-346 · AEAD

1 Introduction

Authenticated Encryption (AE) is a cryptographic technique that combines encryption and message authentication codes (MACs) to provide both confidentiality and integrity for data. It ensures that not only is the information kept secret from unauthorized parties, but also that it has not been tampered with during transit. AEGIS, proposed by Wu and Preneel [WP13a], is one such scheme and its variant AEGIS-128 emerged as one of the winning candidates of the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) [Cae19] for high performance computing applications.

The traditional focus of designers in authenticated encryption with associated data (AEAD) has been on ensuring the security aspects of confidentiality and ciphertext integrity. However, it has been observed in recent years that the previously established notions of confidentiality and integrity may not suffice in various contexts. Among the additional

Received: 2023-11-23 Accepted: 2024-01-23 Published: 2024-03-01

properties explored is the concept of *key commitment*, an area that has received relatively less attention.

Key commitment assures that a ciphertext C can only be decrypted using the same key that was originally used to derive C from some plaintext. Schemes that allow finding a ciphertext that decrypts to valid plaintexts under two different keys do not adhere to the principle of key commitment. The issue of non-key-committing AEAD was initially highlighted in scenarios such as moderation within encrypted messaging [DGRW18, GLR17]. Subsequently, it surfaced in various applications including password-based encryption [LGR21], password-based key exchange [LGR21], key rotation schemes [ADG+22], and envelope encryption [ADG+22].

In even more recent times, there have been new propositions [CR22, BH22] introducing definitions that focus on committing to not only the key, but also the associated data and nonce. Although there have been suggestions for novel schemes [CR22, ADG+22] that align with these diverse definitions, uncertainties persist regarding which existing AEAD schemes actually implement this commitment, and in what manner. Furthermore, several crucial and widely-used AEAD schemes lack demonstrated commitment results. Recently, commitment attacks have been mounted on several widely deployed AEAD schemes, like CCM, GCM, OCB3, etc. [MLGR23].

Contributions. In this work, we assess the key committing security of AEGIS (all its variants) and the other similar AEAD scheme Rocca-S.

A recent assertion has been made suggesting that there are no known attacks on AEGIS in the key committing settings [DL] and AEGIS qualifies as a fully committing AEAD scheme [MST23a, MST23b]. The challenge of attacking the key committing security of AEGIS is also acknowledged as an open problem in [Kö22]. In [DL], it is claimed that finding a collision on a 128-bit tag for variants of AEGIS requires about 2^{64} computations, while for a 256-bit tag, it requires 2^{128} computations. These claims are made under the assumption that AEGIS is fully committing. However, contrary to all these claims, we demonstrate the ability to execute a key committing attack within the FROB game setting [FOR17], which is known to be one of the most stringent key committing frameworks. Thus, we are able to find collisions on tags with a complexity of $O(1)$. This implies that our attacks are also valid in other, more relaxed frameworks, such as CMT-1, CMT-4, and so forth. We also demonstrate a key committing attack against Rocca-S with a complexity of 2^{64}. Note that the previous IETF version of Rocca-S claimed key committing security [NFI23a] while the current IETF version [NFI23b] and conference version of Rocca-S [ABC+23] do not claim any such security. The attacks presented in this work, along with their respective complexities, are summarized in Table 1.

All our attacks exploit the processing of the associated data (AD) as follows. We choose a nonce and key K, N. After the initialization phase of the mode, the internal state becomes a known but uncontrolled value S. We show that, by choosing an appropriate sequence of AD blocks, we can bring the internal state to a fixed value; or at least, a partially fixed value, therefore making collisions immediate or more probable. Once such a collision for a pair $((K_1, N_1, AD_1), (K_2, N_2, AD_2))$ is obtained, we can encrypt any message M and the corresponding ciphertext and tag will have a valid decryption both by K_1 and K_2. Note that the pairs $(K_1, N_1), (K_2, N_2)$ can be arbitrary, and in particular we can have $N_1 \neq N_2$.

Additionally, we show that our techniques applied to Rocca and Tiaoxin-346 do not lead to an attack compromising their key-commitment security. Note that the key committing security of these two schemes still remains an open question. However, the robustness of these schemes against these attacks provides valuable insights into the design considerations for the round update function in such AES-based AEAD schemes.

Table 1: Comparisons of the proposed attack complexities with their generic complexities

AEAD Scheme	Tag Size (bits)	Generic Attack Complexity	Attack Complexity	Reference
AEGIS-128	128	2^{64}	1	Sec. 3.2
AEGIS-256				
AEGIS-128L	128/256	$2^{64}/2^{128}$		
Rocca-S	256	2^{128}	2^{64}	Sec. 3.3

Structure of the Paper. Rest of the paper is organized as follows. In Section 2, we introduce the notions of AEAD and key committing security, and describe the essential features of the schemes that will be attacked. In Section 3, we describe the attacks on AEGIS and Rocca-S. Section 4 illustrates the resistance of Rocca and Tiaoxin-346 against these attacks and provides insights regarding the secure design of such schemes. Since the attacks on AEGIS are easy to verify experimentally, we provide attack vectors in Section A.

2 Preliminaries

2.1 Committing Authenticated Encryption (AE) Frameworks

Consider a symmetric encryption scheme Σ consisting of encryption and decryption algorithms denoted by Σ_{Enc} and Σ_{Dec}, respectively where

$$\Sigma_{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{AD} \times \mathcal{P} \to \mathcal{C},$$

and

$$\Sigma_{Dec} : \mathcal{K} \times \mathcal{N} \times \mathcal{AD} \times \mathcal{C} \to \mathcal{P} \cup \{\perp\}.$$

Here, $\mathcal{K}$, $\mathcal{N}$, $\mathcal{AD}$, $\mathcal{P}$ and $\mathcal{C}$ refer to the key, nonce, associated data, plaintext/message and ciphertext spaces, respectively. Formally, the above scheme is called as a *nonce based authenticated encryption scheme supporting associated data*, or an nAE scheme.

A committing authenticated encryption (cAE) scheme guarantees the definitive determination of the values of its constituent elements, including the key, nonce, associated data, or message, which are utilized to produce the ciphertext. In the committing AE framework, the adversary tries to construct a ciphertext which can be obtained from two different sets of keys, nonces, associated data and messages. Let, $C_i \leftarrow \Sigma_{Enc}(K_i, N_i, AD_i, P_i)$ where $K_i \in \mathcal{K}$, $N_i \in \mathcal{N}$, $AD_i \in \mathcal{AD}$, $P_i \in \mathcal{P}$ and $C_i \in \mathcal{C}$ for $i \in \{1, 2\}$. The adversary aims to find C_1, C_2 such that $C_1 = C_2$ and $(K_1, N_1, AD_1, P_1) \neq (K_2, N_2, AD_2, P_2)$.

Various notions of committing security framework have been introduced [FOR17, CR22, BH22]. We discuss here some of them. In CMT-1, the ciphertext commits exclusively to the key. In the attack, the adversary must produce $((K_1, N_1, AD_1, P_1), (K_2, N_2, AD_2, P_2))$ such that $K_1 \neq K_2$ and $\Sigma_{Enc}(K_1, N_1, AD_1, P_1) = \Sigma_{Enc}(K_2, N_2, AD_2, P_2)$. CMT-4 relaxes the constraints and allows that the commitment can encompass to any of the inputs of Σ_{Enc}, not just the key. The adversary can breach CMT-4 security by constructing a set $((K_1, N_1, AD_1, P_1), (K_2, N_2, AD_2, P_2))$ such that, $(K_1, N_1, AD_1, P_1) \neq (K_2, N_2, AD_2, P_2)$ and $\Sigma_{Enc}(K_1, N_1, AD_1, P_1) = \Sigma_{Enc}(K_2, N_2, AD_2, P_2)$. Bellare and Hoang introduced CMT-3, which is slightly more restrictive than CMT-4. They replaced the constraint $(K_1, N_1, AD_1, P_1) \neq (K_2, N_2, AD_2, P_2)$ with $(K_1, N_1, AD_1) \neq (K_2, N_2, AD_2)$. The FROB game, initially proposed by Farshim, Orlandi, and Rosie [FOR17] and later adapted to the AEAD setting by Grubbs, Lu, and Ristenpart [GLR17], is even more restrictive. It requires the condition $N_1 = N_2$ in addition to $K_1 \neq K_2$. It has been demonstrated that

FROB ($\mathcal{A}$)

1. $(C, (K_1, N_1, AD_1), (K_2, N_2, AD_2)) \xleftarrow{\$} \mathcal{A}$
2. $P_1 \leftarrow \Sigma_{Dec}(K_1, N_1, AD_1, C)$
3. $P_2 \leftarrow \Sigma_{Dec}(K_2, N_2, AD_2, C)$
4. If $P_1 = \perp$ or $P_2 = \perp$ then
   Return false
5. **If** $\boxed{K_1 = K_2 \text{ or } N_1 \neq N_2}$ **then**
   Return false
6. Return true

(a) FROB Game

CMT-1($\mathcal{A}$)

1. $(C, (K_1, N_1, AD_1), (K_2, N_2, AD_2)) \xleftarrow{\$} \mathcal{A}$
2. $P_1 \leftarrow \Sigma_{Dec}(K_1, N_1, AD_1, C)$
3. $P_2 \leftarrow \Sigma_{Dec}(K_2, N_2, AD_2, C)$
4. If $P_1 = \perp$ or $P_2 = \perp$ then
   Return false
5. **If** $\boxed{K_1 = K_2}$ **then**
   Return false
6. Return true

(b) CMT-1 Game

CMT-3($\mathcal{A}$)

1. $(C, (K_1, N_1, AD_1), (K_2, N_2, AD_2)) \xleftarrow{\$} \mathcal{A}$
2. $P_1 \leftarrow \Sigma_{Dec}(K_1, N_1, AD_1, C)$
3. $P_2 \leftarrow \Sigma_{Dec}(K_2, N_2, AD_2, C)$
4. If $P_1 = \perp$ or $P_2 = \perp$ then
   Return false
5. **If** $\boxed{(K_1, N_1, AD_1) = (K_2, N_2, AD_2)}$
   then Return false
6. Return true

(c) CMT-3 Game

CMT-4($\mathcal{A}$)

1. $(C, (K_1, N_1, AD_1), (K_2, N_2, AD_2)) \xleftarrow{\$} \mathcal{A}$
2. $P_1 \leftarrow \Sigma_{Dec}(K_1, N_1, AD_1, C)$
3. $P_2 \leftarrow \Sigma_{Dec}(K_2, N_2, AD_2, C)$
4. If $P_1 = \perp$ or $P_2 = \perp$ then
   Return false
5. **If** $\boxed{(K_1, N_1, AD_1, P_1) = (K_2, N_2, AD_2, P_2)}$
   then Return false
6. Return true

(d) CMT-4 Game

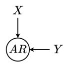
Figure 1: Different Frameworks for Committing Security.

CMT-3 security implies CMT-1, which in turn implies the FROB game [BH22, MLGR23]. In essence, the FROB game presents the most formidable challenge for an adversary to overcome. All the related games are outlined in Fig. 1.

In [CR22], several notions based on the assumptions considered on the key are introduced. Specifically, the authors define the terms *honest*, *revealed*, and *corrupted* keys. A key is deemed *honest* when the adversary possesses no knowledge of it. If the adversary gains knowledge of the key or independently selects a key (i.e., corrupts the key), it is categorized as *revealed* or *corrupted*. Applying this conceptual framework, the attacks discussed in the paper can be contextualized within a *revealed-revealed* scenario, wherein the adversary needs knowledge of both the keys.

2.2 Description of AEGIS

The authenticated encryption scheme with associated data (AEAD) AEGIS was introduced in SAC 2013 [WP13a]. It encompasses three variants: AEGIS-128, AEGIS-256, and AEGIS-128L. In the CAESAR competition [Cae19], AEGIS-128 was selected in the final portfolio for use case 2 (high-performance applications) and AEGIS-128L was a finalist for the same use case.

Across all these variants, the state update function is based on a single round of AES (excluding the key addition operation) denoted as $A(X)$, where X represent a 16-byte state. Specifically, $A(X) = MC \circ SR \circ SB(X)$, where MC, SR, and SB denote the Mixcolumns, Shiftrows, and Subbytes operations, respectively. For more details on these operations refer to [DR00, DR02]. Note that we use the function $AR(X, Y)$ which represents $A(X) \oplus Y$ that is depicted as  in the figures.

State Update. The internal state of AEGIS-128 (resp. AEGIS-256) is made of five (resp. six) 16-byte registers, and the state update function $\texttt{UPDATE}_{A\text{-}128}(S_r, m_r)$ (resp. $\texttt{UPDATE}_{A\text{-}256}(S_r, m_r)$) transforms the internal state S_r to yield the state S_{r+1} and is expressed as:

$$
\begin{aligned}
S_{r+1,0} &= AR(S_{r,b-1}, S_{r,0} \oplus m_r) \\
S_{r+1,1} &= AR(S_{r,0}, S_{r,1}) \\
&\vdots \\
S_{r+1,b-1} &= AR(S_{r,b-2}, S_{r,b-1}) \ ,
\end{aligned}
$$

where $b = 5$ (for AEGIS-128) or 6 (for AEGIS-256), resulting in state sizes of 80 bytes and 96 bytes, respectively. The state update function of AEGIS-128L, denoted as $\texttt{UPDATE}_{A\text{-}128L}(S_r, m_{r,0}, m_{r,1})$, differs slightly from the other two. Let $S_r := S_{r,0}|| \cdots ||S_{r,7}$ be the state after the r-th update, where each $S_{r,i}$ (for $0 \leq i \leq 7$) is a 16-byte block. The function $\texttt{UPDATE}_{A\text{-}128L}(S_r, m_{r,0}, m_{r,1})$ yields the state S_{r+1} where S_{r+1} is defined as follows:

$$
\begin{aligned}
S_{r+1,0} &= AR(S_{r,7}, S_{r,0} \oplus m_{r,0}) \\
S_{r+1,1} &= AR(S_{r,0}, S_{r,1}) \\
S_{r+1,2} &= AR(S_{r,1}, S_{r,2}) \\
S_{r+1,3} &= AR(S_{r,2}, S_{r,3}) \\
S_{r+1,4} &= AR(S_{r,3}, S_{r,4} \oplus m_{r,1}) \\
S_{r+1,5} &= AR(S_{r,4}, S_{r,5}) \\
S_{r+1,6} &= AR(S_{r,5}, S_{r,6}) \\
S_{r+1,7} &= AR(S_{r,6}, S_{r,7}).
\end{aligned}
$$

Algorithm. AEGIS starts with an *initialization phase* where the initial state is loaded with a key K, an initialization vector (IV) IV, and some constants. For AEGIS-128 and AEGIS-128L, the sizes of K and IV are 128 bits, while for AEGIS-256, they are 256 bits. For AEGIS-128, the state is updated using $\texttt{UPDATE}_{A\text{-}128}(S_r, m_r)$ (for $0 \leq r \leq 9$) where each m_r is formed using either K or $K \oplus IV$. Similarly, for AEGIS-256 the state is updated using $\texttt{UPDATE}_{A\text{-}256}(S_r, m_r)$ (for $0 \leq r \leq 15$) where each m_r is derived from K and IV. In the case of AEGIS-128L, the state is updated using $\texttt{UPDATE}_{A\text{-}128L}(S_r, IV, K)$ (for $0 \leq r \leq 9$). In all of these state update functions, S_0 is the initial state.

Following this, based on the lengths of the associated data and plaintext, the states undergo further updates. The associated data and plaintext are separated in 128-bit blocks and processed using the state update function. After each step of the state update function, a 128-bit block of associated data/plaintext is processed for AEGIS-128 and AEGIS-256 (for AEGIS-128L, two 128-bit blocks are encrypted at each step). During the processing of the plaintext, ciphertext blocks are also generated. However, the details are omitted as those are not relevant to the current work.

A *finalization phase* follows in which the state update function will be iterated for seven rounds, before generating the tag. These last updates depend on the lengths of the plaintext and associated data, encoded as 64-bit strings, along with a portion of the previous state. All the 128-bit substates of the final state are XOR-ed to obtain the 128-bit tag. For more comprehensive details on AEGIS, please refer to [WP13a, WP13b, WP16].

2.3 Rocca-S

Rocca-S [ABC+23] is an updated version of Rocca [SLN+21] which has been proposed to standardization at the IETF. We refer here to the latest version of the draft standard document [NFI23a].

Rocca-S employs a 256-bit key and a 256-bit nonce. Its internal state is reduced to seven 16-byte blocks, and its tag length is 256 bits. Much like AEGIS and Rocca, it undergoes phases of initialization, associated data processing, encryption, and finalization, all subject to similar operational constraints. The main difference lies in the round update function, denoted as $\texttt{UPDATE}_{RS}(S_r, X_0, X_1)$, responsible for generating S_{r+1}, where S_r represents the output of the r-th round. If $S_r = S_{r,0}||\cdots||S_{r,6}$, where each $S_{r,i}$ (for $0 \leq i \leq 6$) is a 16-byte block, then S_{r+1} is defined as follows:

$$\begin{aligned}
S_{r+1,0} &= S_{r,6} \oplus S_{r,1} \\
S_{r+1,1} &= A(S_{r,0}) \oplus X_0 \\
S_{r+1,2} &= A(S_{r,1}) \oplus S_{r,0} \\
S_{r+1,3} &= A(S_{r,2}) \oplus S_{r,6} \\
S_{r+1,4} &= A(S_{r,3}) \oplus X_1 \\
S_{r+1,5} &= A(S_{r,4}) \oplus S_{r,3} \\
S_{r+1,6} &= A(S_{r,5}) \oplus S_{r,4} \quad .
\end{aligned}$$

Likewise, the generation of the ciphertexts, as well as the details of initialization and finalization phases, are irrelevant to our attack, and we need only to focus on the AD processing phase.

3 Attacks

In this section, we present a broad overview of the state update mechanism employed in constructions such as AEGIS and Rocca-S. Subsequently, we demonstrate attacks that break the key commitment security of both AEGIS and Rocca-S, leveraging insights derived from this generalized perspective.

3.1 Attack Overview

As outlined in Section 2, AEAD schemes like AEGIS and Rocca undergo four phases: initialization, associated data processing, encryption, and finalization, culminating in the generation of the ciphertext-tag as the output. Throughout these phases, the state updating process is influenced by various parameters: key, initialization vector (IV) or *nonce*, associated data (AD) and plaintext. Considering various parameters, the state updating process can be conceptualized as transitions through different internal states, illustrated in Fig. 2.

Let us denote the initial state as IS_0. The initialization phase is dependent on the key K and the initialization vector IV. Hence, the entire state update process during this phase can be represented as a function $\mathcal{U}_{K,IV}$ which transforms the initial state IS_0 into IS_1. Subsequently, $\mathcal{U}_{AD}$ and $\mathcal{U}_P$ modify the internal states IS_1 and IS_2 to IS_2 and IS_3

respectively, based on the associated data AD and plaintext P. Finally, contingent on the lengths of AD and P, $\mathcal{U}_{|P|,|AD|}$ transforms IS_3 into IS_4. The tag is then generated based on IS_4.

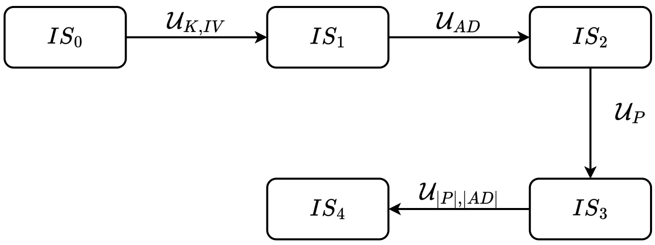


Figure 2: State update as a function of key, initialization vector, associated data and plaintext.

We are specifically interested in analyzing the FROB security. As outlined in Section 2.1, the adversary is required to generate a ciphertext (ciphertext and tag pair) which decrypts to valid plaintexts using two different sets of keys and same IV. Let us consider a set of key, IV, AD, and plaintext, denoted as (K_1, IV_1, AD_1, P_1) which generates a ciphertext-tag pair $C_1||\tau_1$. Consider another key K_2 and an IV IV_2. Note that $K_1 \neq K_2$ and $IV_1 = IV_2$.

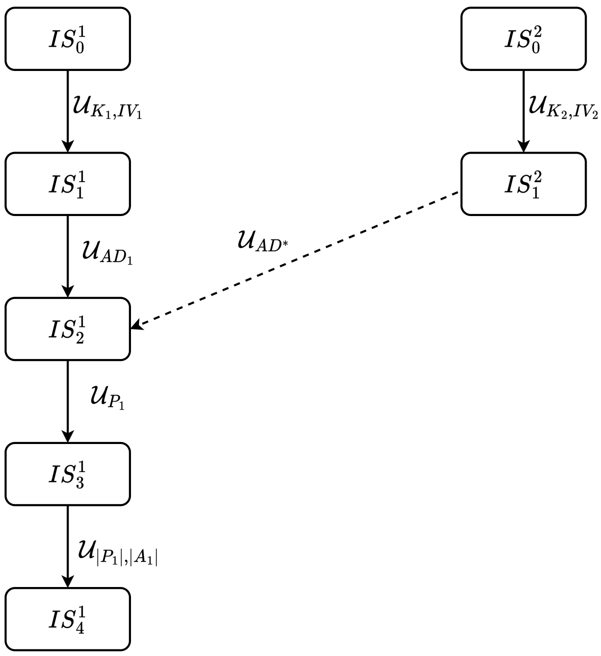


Figure 3: Overview of the attack in FROB framework.

As depicted in Fig. 3, we need to find a AD^* such that $\mathcal{U}_{AD^*}$ transforms IS_1^2 to IS_2^1. If $|AD^*| = |AD_1|$ (the plaintext is P_1), the final state IS_4^1 can be obtained which results in generating the ciphertext-tag pair $C_1||\tau_1$. Consequently, the tuples (K_1, IV_1, AD_1, P_1) and (K_2, IV_2, AD^*, P_1) yield the same ciphertext-tag pair, thereby compromising the FROB security of AEGIS. Hence, the adversary is required to find an AD^* such that $|AD^*| = |AD_1|$. An attack is deemed valid if its complexity is lower than the generic attack complexity. The generic attack for these schemes depends only on the tag length. Indeed, forging a valid tag is sufficient to break the key committing security. Specifically, if tag check is valid, detecting an incorrect key becomes impossible. Thus, for an AEAD scheme with a t-bit tag, the data complexity of a generic attack is $2^{t/2}$. Therefore, any attack that successfully recovers a valid AD^* with a data complexity lower than $2^{t/2}$ can be considered valid.

It is important to note that, following the framework introduced in [CR22], the envisaged attack aligns with the *revealed-revealed* scenario. In this context, the adversary leverages its knowledge of both keys, K_1 and K_2, to uncover the internal states IS_2^1 and IS_1^2, respectively. Subsequently, the adversary identifies an appropriate AD^* which, in turn, breaks the FROB security.

Our attacks are reminiscent of previous works regarding the nonce-misuse (in)security of AEGIS and related AEAD modes (see e.g. [KEM17]). In both cases, the adversary uses successive AD or message words to control different state words. The novelty in our case is that we want to fix the internal state instead of recovering it.

3.2 Attacks on AEGIS

Here, we analyze the FROB security of AEGIS. We show how to find a ciphertext-tag pair which can be decrypted using two different sets of key and nonce. In particular, our focus is on finding (K_1, IV) and (K_2, IV) pairs that produce identical ciphertext-tags when employed with some associated data and plaintexts. AEGIS also follows the generalized state updating process described using Fig. 2. Initially, two keys K_1, K_2 and an initialization vector IV are chosen. Consider that encryption of associated data AD_1 and plaintext P_1 using K_1, IV yields ciphertext-tag $C||\tau$. Let $T = T_0||T_1||T_2||T_3||T_4$ be the internal state after the processing of the AD. Let $S_0 = S_{0,0}||S_{0,1}||S_{0,2}||S_{0,3}||S_{0,4}$ be the internal state after processing of K_2 and IV, and before the AD. We are interested in finding a suitable AD AD^* such that $\mathcal{U}_{AD^*}$ transforms S_0 to T.

Recovering the AD^* for AEGIS-128. With reference to the discussion in Section 3.1 and Fig. 3, the states $S_{0,0}||S_{0,1}||S_{0,2}||S_{0,3}||S_{0,4}$ and $T = T_0||T_1||T_2||T_3||T_4$ take the roles of IS_1^2 and IS_2^1, respectively.

Let $AD^* = AD_0^*||AD_1^*||AD_2^*||AD_3^*||AD_4^*$, where each AD_j^* (for $0 \le j \le 4$) is a 16-byte block. It is quite evident that each T_i can be expressed in terms of AD_i^* and $S_{0,i}$ (for $i \in \{0, 4\}$) as shown below.

$$
\begin{aligned}
T_0 = A(&A(A(A(A(S_{0,0}) \oplus S_{0,1}) \oplus A(S_{0,1}) \oplus S_{0,2}) \\
&\oplus A(A(S_{0,1}) \oplus S_{0,2}) \oplus A(S_{0,2}) \oplus S_{0,3}) \\
&\oplus A(A(A(S_{0,1}) \oplus S_{0,2}) \oplus A(S_{0,2}) \oplus S_{0,3}) \\
&\oplus A(A(S_{0,2}) \oplus S_{0,3}) \oplus A(S_{0,3}) \oplus S_{0,4}) \\
&\oplus AD_4^* \oplus A(A(A(A(S_{0,1}) \oplus S_{0,2}) \oplus A(S_{0,2}) \oplus S_{0,3}) \\
&\oplus A(A(S_{0,2}) \oplus S_{0,3}) \oplus A(S_{0,3}) \oplus S_{0,4}) \\
&\oplus AD_3^* \oplus A(A(A(S_{0,2}) \oplus S_{0,3}) \oplus A(S_{0,3}) \oplus S_{0,4}) \\
&\oplus AD_2^* \oplus A(A(S_{0,3}) \oplus S_{0,4}) \oplus AD_1^* \oplus A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}
\end{aligned}
$$

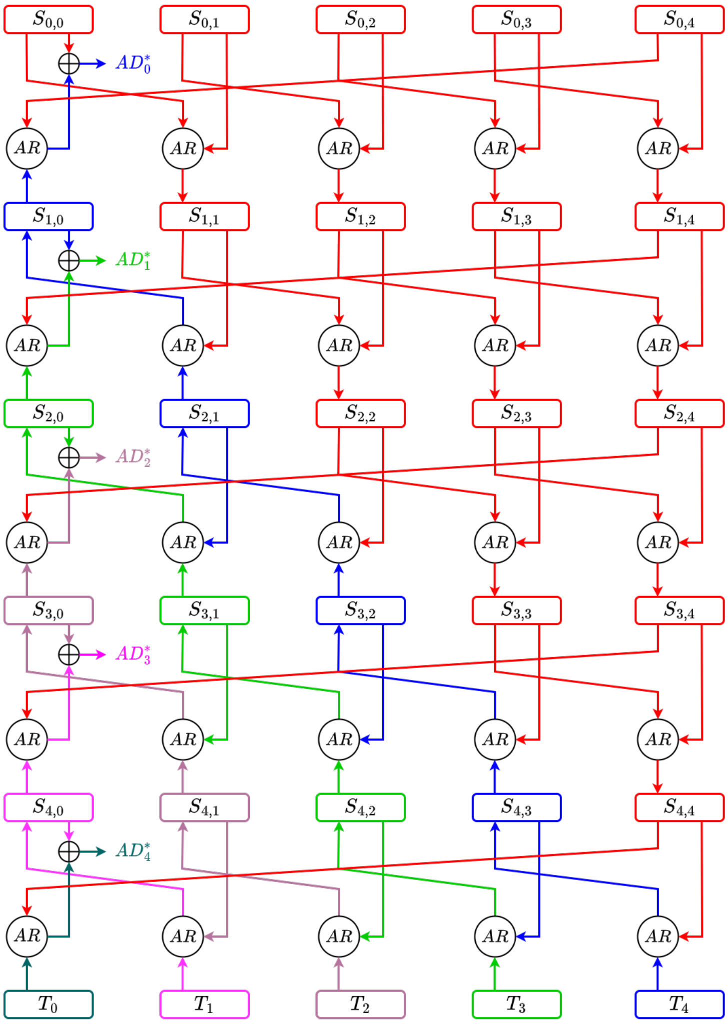


Figure 4: Attack on AEGIS-128

$$
\begin{aligned}
T_1 = {} & A(A(A(A(A(S_{0,1}) \oplus S_{0,2}) \oplus A(S_{0,2}) \oplus S_{0,3}) \\
& \oplus A(A(S_{0,2}) \oplus S_{0,3}) \oplus A(S_{0,3}) \oplus S_{0,4}) \\
& \oplus AD_3^* \oplus A(A(A(S_{0,2}) \oplus S_{0,3}) \oplus A(S_{0,3}) \oplus S_{0,4}) \\
& \oplus AD_2^* \oplus A(A(S_{0,3}) \oplus S_{0,4}) \oplus AD_1^* \oplus A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \\
& \oplus A(A(A(A(S_{0,2}) \oplus S_{0,3}) \oplus A(S_{0,3}) \oplus S_{0,4}) \\
& \oplus AD_2^* \oplus A(A(S_{0,3}) \oplus S_{0,4}) \oplus AD_1^* \oplus A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \\
& \oplus A(A(A(S_{0,3}) \oplus S_{0,4}) \oplus AD_1^* \oplus A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \\
& \oplus A(A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \oplus A(S_{0,0}) \oplus S_{0,1}
\end{aligned}
$$

$$\begin{aligned}
T_2 = A(&A(A(A(A(S_{0,2}) \oplus S_{0,3}) \oplus A(S_{0,3}) \oplus S_{0,4}) \\
&\oplus AD_2^* \oplus A(A(S_{0,3}) \oplus S_{0,4}) \oplus AD_1^* \oplus A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \\
&\oplus A(A(A(S_{0,3}) \oplus S_{0,4}) \oplus AD_1^* \oplus A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \\
&\oplus A(A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \oplus A(S_{0,0}) \oplus S_{0,1}) \\
&\oplus A(A(A(A(S_{0,3}) \oplus S_{0,4}) \oplus AD_1^* \oplus A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \\
&\oplus A(A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \oplus A(S_{0,0}) \oplus S_{0,1}) \\
&\oplus A(A(A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \oplus A(S_{0,0}) \oplus S_{0,1}) \\
&\oplus A(A(S_{0,0}) \oplus S_{0,1}) \oplus A(S_{0,1}) \oplus S_{0,2}
\end{aligned}$$

$$\begin{aligned}
T_3 = A(&A(A(A(A(S_{0,3}) \oplus S_{0,4}) \oplus AD_1^* \oplus A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \\
&\oplus A(A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \oplus A(S_{0,0}) \oplus S_{0,1}) \\
&\oplus A(A(A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \oplus A(S_{0,0}) \oplus S_{0,1}) \\
&\oplus A(A(S_{0,0}) \oplus S_{0,1}) \oplus A(S_{0,1}) \oplus S_{0,2}) \\
&\oplus A(A(A(A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \oplus A(S_{0,0}) \oplus S_{0,1}) \\
&\oplus A(A(S_{0,0}) \oplus S_{0,1}) \oplus A(S_{0,1}) \oplus S_{0,2}) \\
&\oplus A(A(A(S_{0,0}) \oplus S_{0,1}) \oplus A(S_{0,1}) \oplus S_{0,2}) \\
&\oplus A(A(S_{0,1}) \oplus S_{0,2}) \oplus A(S_{0,2}) \oplus S_{0,3}) \\
&\oplus A(A(S_{0,2}) \oplus S_{0,3}) \oplus A(S_{0,3}) \oplus S_{0,4}
\end{aligned}$$

$$\begin{aligned}
T_4 = A(&A(A(A(A(S_{0,4}) \oplus AD_0^* \oplus S_{0,0}) \oplus A(S_{0,0}) \oplus S_{0,1}) \\
&\oplus A(A(S_{0,0}) \oplus S_{0,1}) \oplus A(S_{0,1}) \oplus S_{0,2}) \\
&\oplus A(A(A(S_{0,0}) \oplus S_{0,1}) \oplus A(S_{0,1}) \oplus S_{0,2}) \\
&\oplus A(A(S_{0,1}) \oplus S_{0,2}) \oplus A(S_{0,2}) \oplus S_{0,3}) \\
&\oplus A(A(A(A(S_{0,0}) \oplus S_{0,1}) \oplus A(S_{0,1}) \oplus S_{0,2}) \\
&\oplus A(A(S_{0,1}) \oplus S_{0,2}) \oplus A(S_{0,2}) \oplus S_{0,3}) \\
&\oplus A(A(A(S_{0,1}) \oplus S_{0,2}) \oplus A(S_{0,2}) \oplus S_{0,3}) \\
&\oplus A(A(S_{0,2}) \oplus S_{0,3}) \oplus A(S_{0,3}) \oplus S_{0,4})
\end{aligned}$$

In these equations, the only unknowns are $AD_0^*, \cdots, AD_4^*$. Notably, from the expression for T_4, AD_0^* can be directly recovered. Subsequently, the expression for T_3 involves only AD_0^* and AD_1^* as unknowns. Consequently, after determining AD_0^*, AD_1^* can be deduced from this expression. Following this pattern, the remaining AD_i^*'s can be successively recovered from the corresponding equations, ultimately determining AD^* in constant time.

Refer to Fig. 4 for the overview of the attack. Based on the values of the substates $S_{0,0}, \cdots, S_{0,4}$, some of the internal substates values can be fixed (indicated by the red rectangles in Fig. 4). Notably, when the value of T_4 is set, it deterministically establishes the internal substates $S_{1,0}$ (illustrated by the blue rectangles in Fig. 4). Following a similar approach, the remaining substates of AD^* can be deduced based on the values of the substates of T, as depicted in the figure using various colors.

Recovering AD^* for AEGIS-256/AEGIS-128L. To recover AD^* for both AEGIS-256 and AEGIS-128L, a strategy analogous to the one employed for AEGIS-128 can be applied.

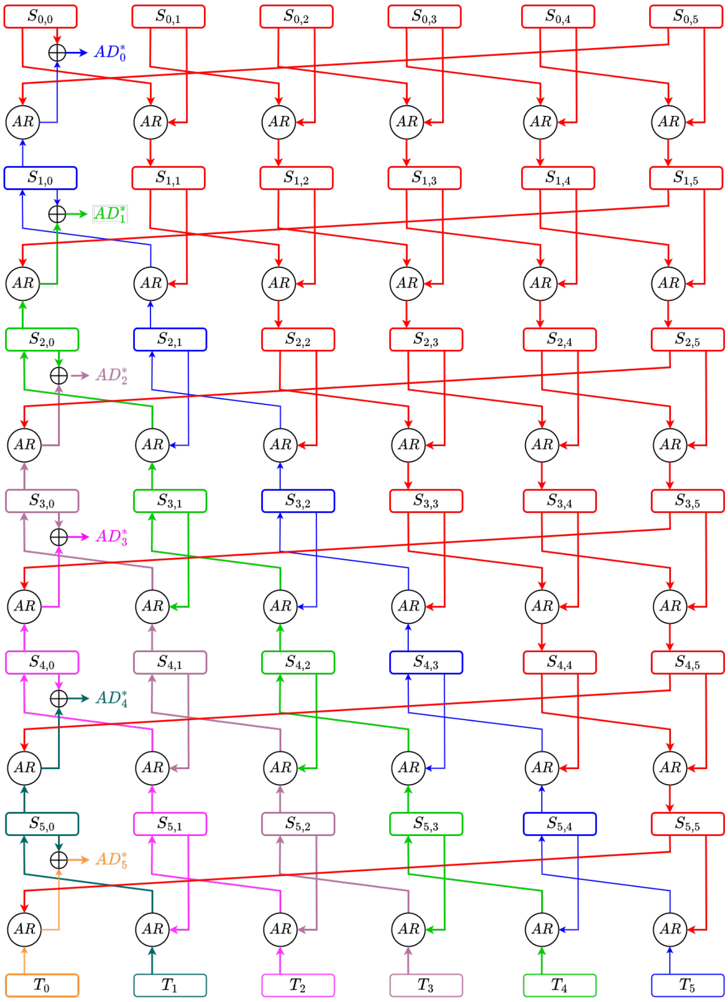


Figure 5: Attack on AEGIS-256

While we omit the detailed equations here (similar to those presented for AEGIS-128), the same technique enables the deterministic recovery of all 128-bit substates of AD^*. The attack strategies for AEGIS-256 and AEGIS-128L are outlined in Fig. 5 and Fig. 6, respectively.

Experimental Verification. In order to verify the validity of our proposed strategy, we have implemented the attacks that break the FROB security. We have provided examples of attack vectors corresponding to the attack on AEGIS-128, AEGIS-256 and AEGIS-128L in Appendix A.1, A.2 and A.3, respectively.

3.3 Attack on Rocca-S

The primary attacking strategy on Rocca-S aligns with the generalized approach outlined in Section 3.1. However, we are not able to control all the internal state blocks, so the complexity is higher and the attack is non-deterministic. For Rocca-S, consider the scenario where a 256-bit key K_1, a 256-bit nonce N, a 6×128-bit $= 768$-bit associated data AD_1, and a plaintext P_1 (of arbitrary length) produce a ciphertext-tag pair $C_1||\tau_1$.

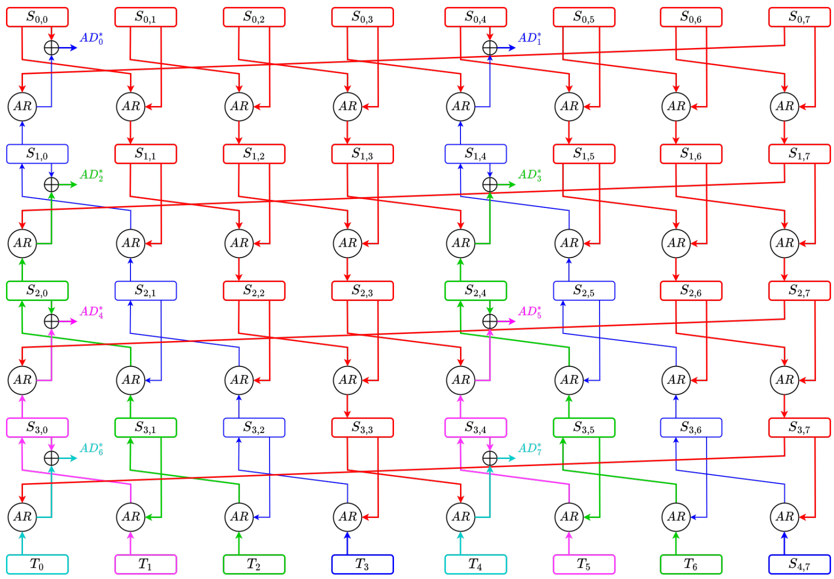


Figure 6: Attack on AEGIS-128L

Assuming an initial state IS_0^1 is established through the initialization process using K_1 and N, applying $\texttt{UPDATE}_{RS}(IS_0^1, Z_0, Z_1)$ for 20 iterations transforms the internal state to IS_1^1. Subsequently, the state undergoes further transformation to IS_2^1 and IS_3^1 after incorporating AD_1 and P_1. Similarly, for another key K_2 and same nonce N, an initial state IS_0^2 is transformed into IS_1^2. The objective is now to identify associated data AD^* that, when applied, can transition the internal state from IS_1^2 to IS_2^1. It is worth to note that the length of AD^*, denoted as $|AD^*|$, must match that of AD_1. This constraint ensures the ability to generate IS_4^1 using a different set of K_2 and N.

Recovering AD^* for Rocca-S. In recovering AD^*, we follow a strategy similar to the one used for attacks on AEGIS. The procedure for recovering AD^* is depicted in Fig. 7. Note that in the figure, the states S_0 and T corresponds to IS_1^2 and IS_2^1, respectively. It is quite evident that the substates of T can be expressed in terms of substates of S_0 and AD^* as follows:

$$
\begin{aligned}
T_0 &= AD_2^* \oplus A(S_{i,1} \oplus S_{i,6}) \oplus A(A(S_{i,4}) \oplus S_{i,3}) \oplus AD_1^* \oplus A(S_{i,3})\\
T_1 &= AD_4^* \oplus A(AD_0^* \oplus A(S_{i,0}) \oplus A(S_{i,5}) \oplus S_{i,4})\\
T_2 &= A(AD_2^* \oplus A(S_{i,1} \oplus S_{i,6})) \oplus AD_0^* \oplus A(S_{i,0}) \oplus A(S_{i,5}) \oplus S_{i,4}\\
T_3 &= A(A(AD_0^* \oplus A(S_{i,0})) \oplus S_{i,1} \oplus S_{i,6}) \oplus A(A(S_{i,4}) \oplus S_{i,3}) \oplus AD_1^* \oplus A(S_{i,3})\\
T_4 &= AD_5^* \oplus A(A(A(S_{i,1}) \oplus S_{i,0}) \oplus A(S_{i,5}) \oplus S_{i,4})\\
T_5 &= A(AD_3^* \oplus A(A(S_{i,2}) \oplus S_{i,6})) \oplus A(A(S_{i,1}) \oplus S_{i,0}) \oplus A(S_{i,5}) \oplus S_{i,4}\\
T_6 &= A(A(AD_1^* \oplus A(S_{i,3})) \oplus A(S_{i,2}) \oplus S_{i,6}) \oplus AD_3^* \oplus A(A(S_{i,2}) \oplus S_{i,6})
\end{aligned}
$$

The values of AD_5^*, AD_3^*, AD_1^*, AD_2^*, AD_0^* and AD_4^* can be recovered successively from the equations pertaining to T_4, T_5, T_6, T_0, T_3 and T_1, respectively. The recovery process is also illustrated in Fig. 7. It should be noted that the substate T_2 cannot be controlled. Therefore, the actual attack on Rocca-S will use 2^{64} values for AD_1 and compute 2^{64} values for AD^* so that with high probability, a collision can be found between them.

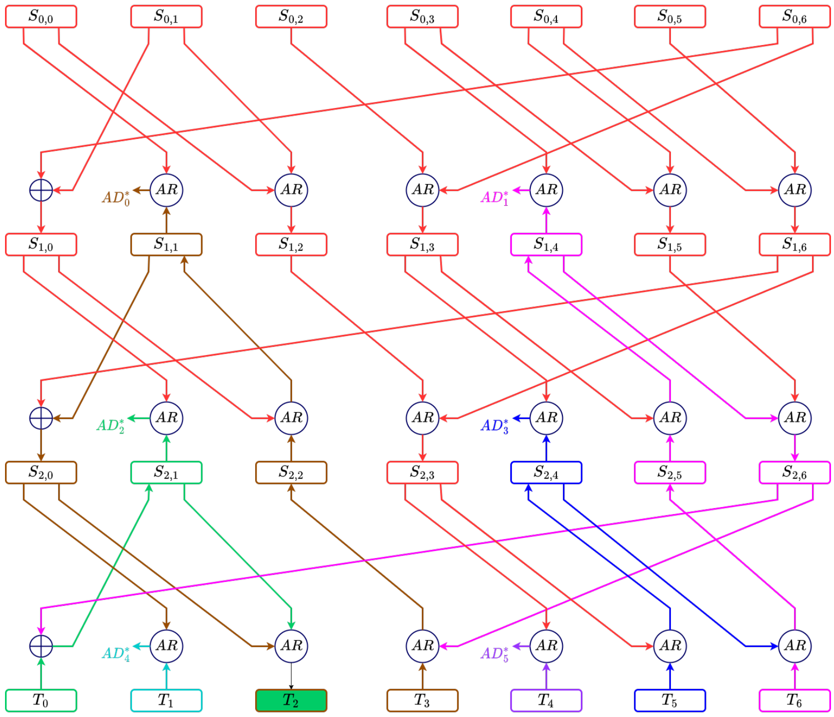


Figure 7: Recovery of AD^* for Rocca-S.

4 Ineffectiveness and Insights: Attacks on Tiaoxin-346 and Rocca

Here, first of all, we discuss about the effect of our attack technique on Tiaoxin-346 and Rocca. Then we discuss about possible countermeasures that arise from some distinction between the round functions of several designs.

4.1 Application on Tiaoxin-346

First, we give a brief overview of Tiaoxin-346. Then, we show that using the proposed technique, the key committing security of Tiaoxin-346 cannot be broken. The fundamental issue is that we lack freedom to control the blocks of internal state. Since too many blocks remain uncontrolled the complexity will remain above the generic attack.

Brief description on Tiaoxin-346. Tiaoxin-346 [Nik16], introduced in the CAESAR competition [Cae19], is a stream cipher based design and composed of four phases- initialization, associated data processing, encryption and finalization.

The Tiaoxin-346 state is composed of thirteen 128-bit words divided in three separate registers. If the state after r-th round S_r is denoted using 128-bit substates as

$$(U_{r,0}, U_{r,1}, U_{r,2}, V_{r,0}, V_{r,1}, V_{r,2}, V_{r,3}, W_{r,0}, W_{r,1}, W_{r,2}, W_{r,3}, W_{r,4}, W_{r,5}),$$

then the round update function $\texttt{UPDATE}_T(S_r, X_0, X_1, X_2)$ that is used to generate S_{r+1} can be formalized as follows:

$$
\begin{aligned}
U_{r+1,0} &= U_{r,0} \oplus X_0 \oplus A(U_{r,2}) \\
U_{r+1,1} &= A(U_{r,0}) \\
U_{r+1,2} &= U_{r,1} \\
V_{r+1,0} &= V_{r,0} \oplus X_1 \oplus A(V_{r,3}) \\
V_{r+1,1} &= A(V_{r,0}) \\
V_{r+1,i} &= V_{r,i-1} \text{ (for } i \in \{2,3\}) \\
W_{r+1,0} &= W_{r,0} \oplus X_2 \oplus A(W_{r,5}) \\
W_{r+1,1} &= A(W_{r,0}) \\
W_{r+1,i} &= W_{r,i-1} \text{ (for } i \in \{2,3,4,5\})
\end{aligned}
$$

In the initialization phase, the state S_0 is initialized using a nonce, a secret key and some constants Z_0 and Z_1. Then, S_0 is updated using $\texttt{UPDATE}_T(S_0, Z_0, Z_1, Z_0)$ to obtain the state S_{15}. The associated data AD is divided into 128-bit words $AD_0||AD_1||\cdots||AD_{2d+1}$ (padding bits are added to make the AD length a multiple of 256). Then the function $\texttt{UPDATE}_T(S_{15+i}, AD_{2i}, AD_{2i+1}, AD_i \oplus AD_{2i+1})$ is called for $0 \le i \le d$ to obtain the final state S_{r+d+1}. Similarly, in the encryption phase, two 128-bit words from the plaintext P is used to update the state in each round. In the finalization phase, the length of AD and P in terms of number of bits is used to update the state. For details on the Tiaoxin-346, refer to [Nik16].

Attack Idea. Refer to Fig. 8 for the proposed attack technique on Tiaoxin-346. Note that, following Section 3.1, the states $U_{0,0}||\cdots||U_{0,2}||V_{0,0}||\cdots||V_{0,3}||W_{0,0}||\cdots||W_{0,5}$ and $T_{u,0}||\cdots||T_{u,2}||T_{v,0}||\cdots||T_{v,3}||T_{w,0}||\cdots||T_{w,5}$ corresponds to IS_1^2 and IS_2^1, respectively. We are interested in recovering an associated data AD^* such that $\mathcal{U}_{AD^*}$ transforms IS_1^2 to IS_2^1.

As shown in the figure, each $T_{u,i}$ controls the value of c_{5-i} for $0 \le i \le 5$. Hence, the values of c_i's can be determined in a constant time. In the second register, the values of b_5 and b_4 can be controlled freely. The remaining b_i's ($i \in \{0,1,2,3\}$) are determined by the $T_{u,i}$, b_4 and b_5. Similarly, for the first register, the values for a_3, a_4 and a_5 can be freely chosen.

In the processing of associated data in Tiaoxin-346, each c_i should be equal to $a_i \oplus b_i$. For $i \in \{3,4,5\}$, a_i's are chosen such that $a_i = b_i \oplus c_i$. However, a_0, a_1 and a_2 cannot be controlled freely and thus the condition $c_i = a_i \oplus b_i$ is satisfied for $0 \le i \le 2$ with regards to three 128-bit collisions. Hence, it is expected to find a valid AD^* using $2^{64} \times 2^{64} \times 2^{64} = 2^{192}$ different iterations of AD_1. This attack complexity is worse than the generic attack on Tiaoxin-346 as it uses a 128-bit tag, i. e., the generic collision probability is 2^{64}.

4.2 Application on Rocca

Here, we make a similar observation on Rocca. We provide a brief description of the scheme and illustrate how our technique can (not) be employed.

Brief description on Rocca. Rocca [SLN+21, SLN+22] is an AES-based AEAD scheme specifically designed for 6G applications. Its internal state contains eight 16-byte blocks and its state update function $\texttt{UPDATE}_R()$ relies also on the AES round function A. More precisely, it accepts two additional 16-byte blocks X_0, X_1 which can be constants or message / AD blocks, and modifies the state accordingly. Let $S_r := S_{r,0}||\cdots||S_{r,7}$ be the state after the r-th update, where $S_{r,i}$ ($0 \le i \le 7$) are the 128-bit substates. Then

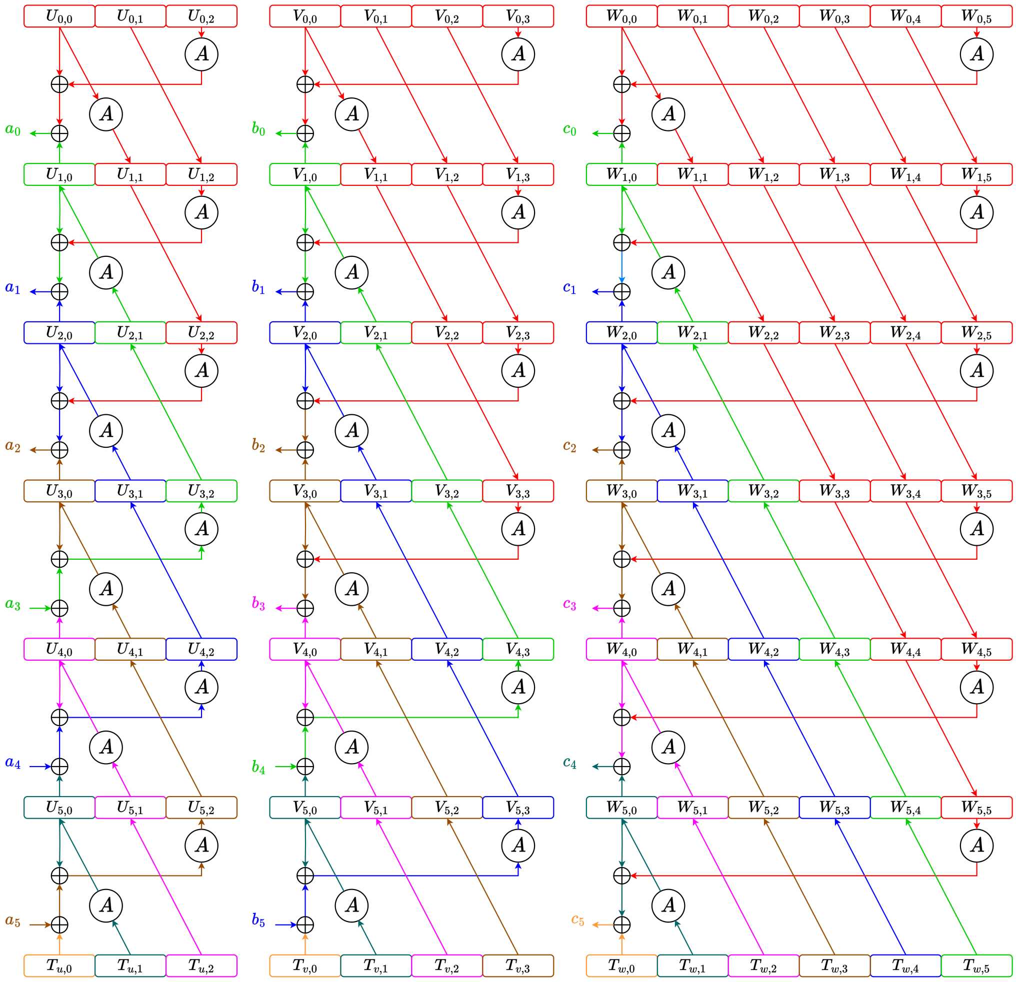


Figure 8: Attack Overview on Tiaoxin-346

$S_{r+1} = \texttt{UPDATE}_R(S_r, X_0, X_1)$ is defined as follows:

$$\begin{aligned}
S_{r+1,0} &= S_{r,7} \oplus X_0 \\
S_{r+1,1} &= A(S_{r,7}) \oplus S_{r,0} \\
S_{r+1,2} &= S_{r,1} \oplus S_{r,6} \\
S_{r+1,3} &= A(S_{r,1}) \oplus S_{r,2} \\
S_{r+1,4} &= S_{r,3} \oplus X_1 \\
S_{r+1,5} &= A(S_{r,3}) \oplus S_{r,4} \\
S_{r+1,6} &= A(S_{r,4}) \oplus S_{r,5} \\
S_{r+1,7} &= S_{r,6} \oplus S_{r,0}
\end{aligned}$$

Algorithm. Like for AEGIS, we omit details which are irrelevant to our attack, as we mostly need to focus on the absorption of the AD blocks during the AD processing phase.

Rocca starts with an initialization phase where the state S_0 is initialized by loading a 256-bit key $K_0||K_1$, a 128-bit nonce N, and two 128-bit constants Z_0, Z_1, along with additional constants. The operation $\texttt{UPDATE}_R(S_i, Z_0, Z_1)$ is iteratively executed for $0 \leq i \leq 19$ to compute the state S_{20}. When processing the associated data AD, padding bits are appended to form AD^* in such a way that its length, measured in bits, is a multiple

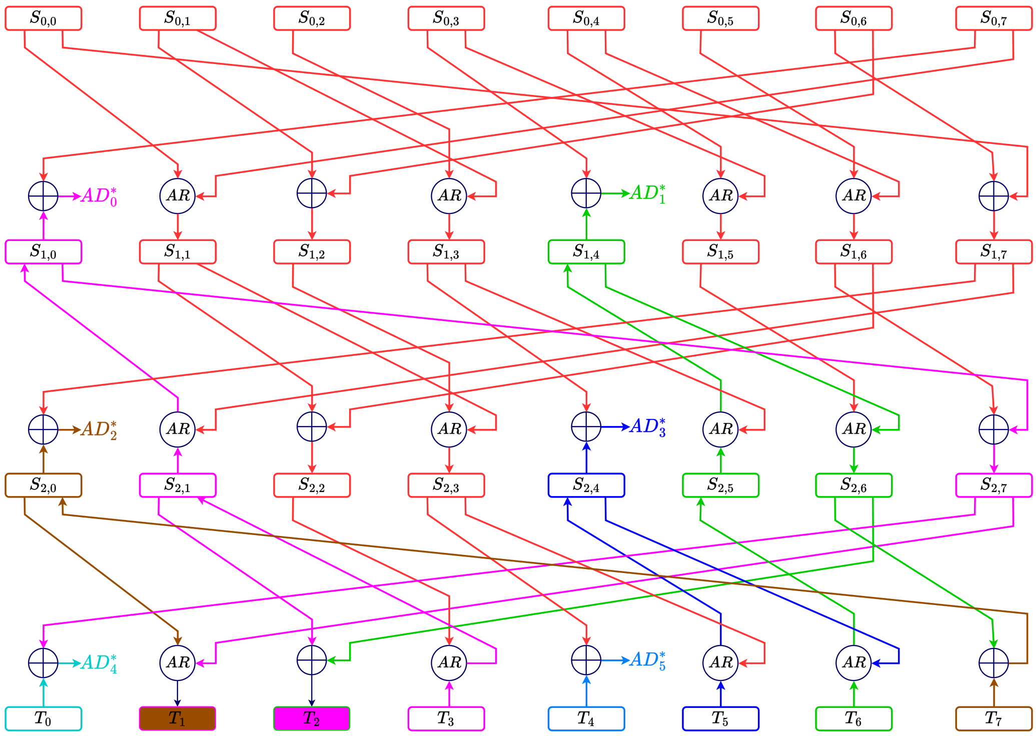


Figure 9: Recovering AD^* for Rocca. Note that $AD^* = AD_0^*||\cdots||AD_5^*$.

of 256. The operation $\texttt{UPDATE}_R(S_{20+i}, AD_{2i}^*, AD_{2i+1}^*)$ is executed for $0 \le i \le d$, where $AD^* = AD_0^*||AD_1^*||\cdots||AD_{2d+1}^*$. The plaintext P is processed similarly, except that it also intervenes in the computation of (pairs of) ciphertext blocks which are returned. In the finalization step, the state update is called with the binary-encoded lengths of AD and P are used, and the 128-bit tag is computed as the XOR of all state blocks.

Attack Idea. With reference to Section 3.1, we discuss here the process of recovering AD^* for Rocca. Refer to Fig. 9 for the overview of the attack technique. Strategy similar to the one employed for Rocca-S is applied for Rocca. Like Rocca-S, we consider that the states S_0 and T corresponds to IS_1^2 and IS_2^1, respectively and the 128-bit substates of T are expressed in terms of 128-bit substates of S_0 and AD^*.

$$
\begin{aligned}
T_0 &= AD_4^* \oplus AD_0^* \oplus S_{i,7} \oplus A(S_{i,5}) \oplus S_{i,4}\\
T_1 &= A(AD_2^* \oplus S_{i,0} \oplus S_{i,6}) \oplus AD_0^* \oplus S_{i,7} \oplus A(S_{i,5}) \oplus S_{i,4}\\
T_2 &= A(AD_0^* \oplus S_{i,7}) \oplus S_{i,0} \oplus S_{i,6} \oplus A(A(S_{i,4}) \oplus S_{i,3}) \oplus AD_1^* \oplus S_{i,3}\\
T_3 &= A(A(S_{i,0}) \oplus S_{i,7} \oplus A(S_{i,5}) \oplus S_{i,4}) \oplus A(AD_0^* \oplus S_{i,7}) \oplus S_{i,0} \oplus S_{i,6}\\
T_4 &= AD_5^* \oplus A(S_{i,1} \oplus S_{i,6}) \oplus A(S_{i,0}) \oplus S_{i,7}\\
T_5 &= A(AD_3^* \oplus A(S_{i,2}) \oplus S_{i,1}) \oplus A(S_{i,1} \oplus S_{i,6}) \oplus A(S_{i,0}) \oplus S_{i,7}\\
T_6 &= A(A(AD_1^* \oplus S_{i,3}) \oplus A(S_{i,2}) \oplus S_{i,1}) \oplus AD_3^* \oplus A(S_{i,2}) \oplus S_{i,1}\\
T_7 &= AD_2^* \oplus S_{i,0} \oplus S_{i,6} \oplus A(A(S_{i,4}) \oplus S_{i,3}) \oplus AD_1^* \oplus S_{i,3}
\end{aligned}
$$

Note that successively processing the equations for T_3, T_4, T_5, T_6, T_7 and T_0, the values of AD_0^*, AD_5^*, AD_3^*, AD_1^*, AD_2^* and AD_4^* can be determined. As illustrated in the figure, the six distinct 128-bit blocks of AD^* exert control over six out of the eight blocks in T.

However, the remaining two blocks remain beyond control, and a valid solution of AD^* requires collisions on two 128-bit blocks T_1 and T_2. From the arguments pertaining to birthday-bound problem, it is expected that iterating through $2^{64} \times 2^{64} = 2^{128}$ different values of AD_1, a valid AD^* can be recovered. However Rocca has a 128-bit tag and thus the generic attack has a complexity of 2^{64}. Note that similar observation (corresponding to the recovery of AD^*) is also made by Takeuchi and Iwata while mounting a key recovery attack on Rocca [TI].

4.3 Insights into Round Update Function: Resistance against Key Committing Attacks

Here, we delve into the differences among round update functions that resist the attack strategy proposed in this work. Unlike solutions proposed in [ADG+22, BH22, CR22], which employ pseudo-random functions or hash-based approaches to transform a generic AEAD scheme into a key committing one, our discussion focuses on insights into selecting a round function to enhance resistance against key committing attacks.

First, let's look into the design of Tiaoxin-346. The resilience of Tiaoxin-346 is primarily derived from the utilization of three blocks of messages/associated data in each round, with the third block being the XOR sum of the first two blocks. Notably, if the third block of the message is not the XOR of the first two blocks, a deterministic attack becomes feasible. The length of the tag and the size of each register also significantly influence the attack's effectiveness. As discussed in Section 4.1, finding a valid AD^* requires collision on three 128-bit blocks. The size of the register plays a crucial role in determining the overall data complexity of the attack. If the smallest register has m_s 128-bit blocks, then during the AD absorption in the smallest register, m_s blocks of AD cannot be controlled freely. The success of the attack depends on the collision in these m_s blocks, resulting in a complexity of $2^{64}m_s$. The resistance against key committing attacks is achieved if the length of the tag is less than $2^{128}m_s$.

Now, we discuss about the resistance of Rocca against these attacks. Similar to AEGIS-128L, the state update function of Rocca employs a 1024-bit state, and in each round, two 128-bit blocks of messages/associated data are absorbed. However, the application of our technique results in a deterministic attack against AEGIS, whereas for Rocca, it requires a data complexity of 2^{128}. Notably, Rocca achieves full diffusion after 7 rounds, whereas AEGIS-128L requires 10 rounds for full diffusion.

For AEGIS, the messages absorbed in the i-th round have effects on at most four state blocks after the $(i+4)$-th round. Referring to Fig. 6, consider AD_0^* and AD_1^*. After four rounds, AD_0^* affects blocks T_0, T_1, T_2, and T_3, while AD_1^* affects the remaining blocks. These affected blocks are disjoint, facilitating the discovery of a valid AD^* in constant time. In contrast, for Rocca, AD_0^* and AD_1^* affect blocks (T_0, T_1, T_2, T_3) and (T_2, T_6, T_7), respectively (refer to Fig.9). After three rounds of Rocca, due to faster diffusion, at most six out of eight substate blocks can be independently controlled. Increasing the round number does not improve the attack, as two or more AD blocks control the output substates, and recovering a valid AD^* depends on the collision in these blocks. This fast diffusion property of Rocca enhance the security as compared to AEGIS-128.

5 Conclusion

The issue of key commitment security in AEGIS has been a significant and persisting question. This work addresses this gap by conducting a thorough analysis of AEGIS. Our analysis, considering various existing frameworks, culminated in the development of a practical attack applicable to all variants of AEGIS. However, in frameworks where an additional constraint of identical associated data is imposed, the proposed attacks will not be effective. We have also demonstrated that the key committing security of Rocca-S can be compromised by the proposed attacks. These findings emphasize the ongoing importance of research and evaluation in AEAD security, especially within the framework of key commitment. Nevertheless, AEAD schemes such as Rocca and Tiaoxin-346 have proven resistant to the presented attacks. Their immunity to key committing attacks offers valuable insights into the design of these ciphers, which we believe will be instrumental in shaping future AES-based AEAD schemes.

Acknowledgments

We are grateful to the anonymous reviewers of ToSC and Mustafa Khairallah for their detailed comments and suggestions. This research was in part conducted under a contract of "Research and development on new generation cryptography for secure wireless communication services" among "Research and Development for Expansion of Radio Wave Resources (JPJ000254)", which was supported by the Ministry of Internal Affairs and Communications, Japan. This result is obtained from the commissioned research (JPJ012368C05801) by the National Institute of Information and Communications Technology (NICT), Japan. This work has been supported by the French Agence Nationale de la Recherche through the OREO project under Contract ANR-22-CE39-0015, and through the France 2030 program under grant agreement ANR-22-PECY-0010.

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 *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.

[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.

[Cae19] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. https://competitions.cr.yp.to/caesar-submissions.html, 2019.

[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.

[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.

[DL] Frank Denis and Samuel Lucas. The AEGIS Family of Authenticated Encryption Algorithms. https://datatracker.ietf.org/doc/draft-irtf-cfrg-aegis-aead/04/.

[DR00] Joan Daemen and Vincent Rijmen. Rijndael for AES. In *The Third Advanced Encryption Standard Candidate Conference, April 13-14, 2000, New York, New York, USA*, pages 343–348. National Institute of Standards and Technology,, 2000.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002.

[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.

[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.

[KEM17] Daniel Kales, Maria Eichlseder, and Florian Mendel. Note on the robustness of CAESAR candidates. *IACR Cryptol. ePrint Arch.*, page 1137, 2017.

[Kö22] Stefan Kölbl. Open Questions around Key Committing AEADs. https://frisiacrypt2022.cs.ru.nl/assets/slides/stefan-frisiacrypt2022.pdf, 2022.

[LGR21] Julia Len, Paul Grubbs, and Thomas Ristenpart. Partitioning Oracle Attacks. In Michael 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.

[MST23a] John Preuß Mattsson, Ben Smeets, and Erik Thormarker. Proposals for Standardization of Encryption Schemes. https://csrc.nist.gov/csrc/media/Events/2023/third-workshop-on-block-cipher-modes-of-operation/

documents/accepted-papers/Proposals%20for%20Standardization%20of%20Encryption%20Schemes%20Final.pdf, 2023.

[MST23b] John Preuß Mattsson, Ben Smeets, and Erik Thormarker. Proposals for Standardization of Encryption Schemes. https://csrc.nist.gov/csrc/media/Presentations/2023/proposal-for-standardization-of-encryption-schemes/images-media/sess-4-mattsson-bcm-workshop-2023.pdf, 2023.

[NFI23a] Yuto Nakano, Kazuhide Fukushima, and Takanori Isobe. Encryption algorithm Rocca-S. https://datatracker.ietf.org/doc/draft-nakano-rocca-s/03/, 2023.

[NFI23b] Yuto Nakano, Kazuhide Fukushima, and Takanori Isobe. Rocca-S: IETF Version. https://datatracker.ietf.org/doc/draft-nakano-rocca-s/04/, 2023.

[Nik16] Ivica Nikolić. Tiaoxin-346 for the CAESAR Competition. http://competitions.cr.yp.to/round3/tiaoxinv21.pdf, 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.

[TI] Ryunosuke Takeuchi and Tetsu Iwata. Key Recovery Attack against Modified Version of Rocca. Private Communication.

[WP13a] 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.

[WP13b] Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm. Cryptology ePrint Archive, Paper 2013/695, 2013.

[WP16] Hongjun Wu and Bart Preneel. AEGIS: A Fast Authenticated Encryption Algorithm (v1.1). https://competitions.cr.yp.to/round3/aegisv11.pdf, 2016.

A Attack Vectors

Note that, in the attack vectors, we have provided a ciphertext/tag pair. However, the tuple $((K_1, IV_1, AD_1), (K_2, IV_2, AD^*))$ (here $IV_1 = IV_2$) works with any plaintext, i. e., if we encrypt a plaintext with both (K_1, IV_1, AD_1) and (K_2, IV_2, AD^*), it generates same ciphertext/tag pair. In this way, numerous ciphertext/tag pair can be generated which can be decrypted to valid plaintexts.

In the vectors provided, the leftmost bit is the least significant bit (LSB). Consider a 16-bit string $b_0 \cdots b_{15}$ where b_0 is the LSB and b_{15} is the most significant bit (MSB). Using the vectors, the above string is denoted as $[b_0 \cdots b_7 \quad b_8 \cdots b_{15}]$.

A.1 Attack Vector for AEGIS-128

$C\|\|\tau=$	[0xA5	0xA7	0x7C	0x8D	0x8D	0xB5	0xEB	0x88	0x35	0x72
	0x71	0x78	0xDA	0x00	0x15	0xFF	0xBC	0x1D	0xB4	0xF6
	0x28	0x7B	0x96	0xEE	0x1E	0xA0	0xF8	0xEC	0x0C	0xFF
	0x32	0x4B]								

$K_1=$	[0x62	0x1F	0x61	0xFA	0x65	0x84	0x70	0xCC	0x18	0x4B
	0x39	0x45	0x3D	0xAB	0x75	0x80]				

$IV_1=$	[0xCE	0xD7	0xE2	0xF0	0xB2	0xAE	0x0D	0x0D	0x3E	0x82
	0x5F	0xFC	0xE4	0x6F	0xC7	0xCF]				

$AD_1=$	[0xBE	0x17	0x84	0xAA	0x3B	0x98	0x29	0xBC	0xCC	0xF3
	0x81	0x04	0x11	0x57	0x4F	0x43	0xFB	0x86	0xA4	0xE3
	0xD6	0x34	0x1C	0x15	0xB7	0x07	0x8E	0x2C	0x91	0x75
	0x86	0xE2	0x89	0x94	0x5D	0x69	0x85	0x55	0xB0	0xEE
	0x68	0x70	0x27	0x71	0xF1	0x0A	0xF8	0x89	0x30	0xF9
	0x35	0x7B	0x8D	0xFE	0x1F	0x07	0xD1	0x6F	0x39	0xD2
	0x44	0x1D	0xC3	0x83	0x31	0x65	0xAF	0x74	0x55	0x03
	0xA6	0xB3	0xD3	0x2C	0x15	0x8C	0x86	0xA3	0xFA	0xCF]

$K_2=$	[0xFC	0xF9	0x24	0xED	0x84	0x21	0x9B	0xD8	0x24	0xEB
	0x58	0xB9	0x01	0xA8	0x08	0x82]				

$IV_2=$	[0xCE	0xD7	0xE2	0xF0	0xB2	0xAE	0x0D	0x0D	0x3E	0x82
	0x5F	0xFC	0xE4	0x6F	0xC7	0xCF]				

$AD^*=$	[0x15	0x7E	0xC0	0x40	0x64	0xDB	0x40	0x47	0xDC	0xE2
	0x56	0x7D	0x41	0x6C	0x5D	0x08	0x71	0xB4	0xDB	0xD8
	0x76	0xC5	0xCC	0xD1	0x44	0xF0	0x58	0x91	0xF5	0xED
	0x22	0x91	0x3F	0xA8	0xEC	0x97	0x71	0xD5	0xD2	0x7C
	0x28	0xF7	0x53	0xBB	0xE0	0x5A	0xD1	0xBF	0x34	0xF2
	0x44	0x14	0xE7	0x37	0x88	0x61	0xB3	0x0E	0x5C	0x75
	0x61	0x84	0xBE	0x03	0x0F	0xBB	0x57	0xF1	0x3B	0x2D
	0x93	0x74	0xCB	0x70	0x57	0xFC	0x9D	0xF9	0xE4	0x2B]

A.2 Attack Vector for AEGIS-256

$C||\tau=$ [0x5F 0x74 0x00 0x73 0x1E 0x88 0x1D 0x84 0xAE 0x0A
0x18 0xE2 0x16 0x9B 0x6E 0x98 0xB0 0x8D 0x5C 0xB1
0x74 0x9F 0x53 0x80 0xF6 0xE0 0x9B 0x0F 0x33 0x1D
0x42 0xF0]

$K_1=$ [0x15 0x86 0x32 0x3E 0x9C 0x71 0xB4 0x9F 0x13 0x36
0xAC 0x8D 0x7D 0x37 0x1B 0x9B 0x7A 0x80 0x0D 0x63
0x7D 0x27 0x46 0xFF 0x5C 0x55 0x0E 0x5A 0xEC 0xE7
0x8C 0x81]

$IV_1=$ [0xD9 0x9D 0x22 0x35 0x4E 0xF7 0x15 0xF8 0x70 0x88
0xEF 0x8E 0x88 0xBE 0xC0 0x1C 0x6A 0xD7 0xFE 0xDF
0x43 0xF7 0x8D 0x61 0x5D 0x88 0xB9 0x00 0xCA 0x62
0x29 0xF0]

$AD_1=$ [0x8E 0x15 0x9D 0xB0 0x18 0x2E 0x11 0xFC 0x46 0xE0
0x28 0xA6 0x49 0x58 0xC5 0x5E 0xFE 0x77 0x01 0xBA
0x07 0xB6 0x19 0x8C 0x3C 0x1D 0x1E 0xB7 0x63 0x5E
0x97 0xB0 0xCD 0x58 0x06 0x81 0x03 0xD5 0x64 0xDC
0x36 0xA2 0x26 0xCB 0x2B 0xC5 0xE6 0x5E 0x16 0xCB
0xB3 0x19 0xB2 0xFB 0x3C 0x39 0x3B 0x8E 0xCF 0xF1
0x79 0x06 0x61 0x4D 0x67 0xFF 0xF0 0xFB 0x86 0xC5
0x8E 0x61 0x8D 0x74 0x8F 0x52 0x7B 0x0C 0x75 0xC6
0x85 0x84 0x0D 0x09 0xC2 0xCA 0xF1 0xDB 0x18 0xC2
0x43 0x6F 0xE9 0x11 0x37 0x00]

$K_2=$ [0x5C 0xD5 0x0D 0xFB 0x4F 0x8A 0x55 0x31 0x1C 0xF3
0xCC 0xBD 0xF0 0xA4 0xD5 0x80 0x5D 0xAA 0x0B 0x2E
0x98 0xDE 0x8E 0x09 0x1F 0x82 0x04 0xBA 0x39 0x29
0x7C 0x78]

$IV_2=$ [0xD9 0x9D 0x22 0x35 0x4E 0xF7 0x15 0xF8 0x70 0x88
0xEF 0x8E 0x88 0xBE 0xC0 0x1C 0x6A 0xD7 0xFE 0xDF
0x43 0xF7 0x8D 0x61 0x5D 0x88 0xB9 0x00 0xCA 0x62
0x29 0xF0]

$AD^*=$ [0xB9 0x55 0xF7 0x5C 0xB9 0x91 0xC3 0x17 0xD1 0xC4
0x2A 0x7D 0x7C 0x3A 0xC8 0x1E 0x84 0x62 0xF4 0x03
0x69 0x44 0x7F 0x20 0x6E 0xFB 0xF3 0x0E 0xD1 0x47
0x8A 0xD0 0xA4 0xA0 0x0C 0x00 0xA4 0x6B 0x84 0x71
0x14 0x14 0x70 0xF3 0xD3 0x4E 0x88 0xD7 0xF8 0xC3
0xFD 0xAE 0xAA 0x2A 0xA1 0x98 0xFC 0x07 0x87 0x74
0x7C 0x7D 0xBB 0x06 0x5E 0x56 0x1C 0x41 0x67 0x54
0x54 0xDF 0x1F 0x49 0x0A 0x1D 0x9B 0xE0 0x7E 0x05
0xF1 0x41 0xE9 0x2A 0x11 0x0E 0x91 0x87 0xB7 0xBA
0xA8 0x2F 0xBC 0x67 0x2B 0xEF]

A.3 Attack Vector for AEGIS-128L

$C||\tau=$ [0xE2 0xF5 0x27 0xF6 0x7D 0xD5 0xC9 0x77 0x5C 0x0C
0x0A 0x09 0x0C 0x06 0x71 0x5A 0x4F 0x78 0x84 0xF1
0x2F 0x08 0xB8 0xF6 0x05 0xD4 0xED 0x86 0x89 0x52
0x37 0xA0]

$K_1=$ [0x09 0xAA 0x5D 0x16 0x70 0x62 0x2E 0xED 0xFB 0x18
0x8E 0x9D 0x17 0xA9 0x71 0x18]

$IV_1=$ [0x24 0xF2 0xEA 0xAF 0xAE 0xCA 0x95 0xFF 0xC8 0x4A
0x3B 0x94 0x36 0x8C 0xD2 0xC1]

$AD_1=$ [0x92 0x9D 0xBF 0xD2 0x4E 0xAE 0x0A 0x2E 0xAC 0xB1
0x1E 0x0F 0x82 0x28 0x1A 0x2D 0x4B 0x7F 0x15 0xF2
0x32 0x53 0x7B 0xFC 0x00 0xDC 0x98 0x08 0xA8 0xF7
0x57 0xA9 0xB5 0x38 0xF6 0x4E 0x0F 0xD1 0x6F 0x88
0xD1 0x10 0x7D 0xE9 0x11 0x35 0x8C 0x27 0x24 0xDE
0x8E 0x14 0xF5 0x51 0x21 0x0E 0xEB 0x90 0x95 0xB6
0x4A 0xAC 0x7D 0x1D 0xF9 0xAE 0xC5 0xEA 0x99 0x06
0xF5 0x0E 0x57 0x8A 0x8B 0xB5 0x64 0x3C 0x15 0x4C
0xD0 0xC2 0xE3 0xE6 0x76 0x82 0xE6 0xDF 0x63 0xB4
0x30 0x27 0xAE 0x13 0x94 0xD8 0x5D 0x16 0x6A 0x2E
0x3B 0x7C 0x0B 0xB6 0xAA 0xB9 0x98 0x2C 0x03 0x44
0xF0 0x98 0x54 0xB5 0x1A 0xBA 0x37 0xB6 0x51 0x70
0xCC 0xDB 0x91 0xCA 0x36 0x65 0x45 0x08]

$K_2=$ [0x1A 0x69 0x72 0xD1 0x60 0x38 0x0B 0xA9 0xD6 0x0D
0x6A 0xF6 0x1E 0xCB 0xEA 0x75]

$IV_2=$ [0x24 0xF2 0xEA 0xAF 0xAE 0xCA 0x95 0xFF 0xC8 0x4A
0x3B 0x94 0x36 0x8C 0xD2 0xC1]

$AD^*=$ [0xB6 0x58 0x24 0xE0 0x6F 0x0E 0xA4 0x06 0x42 0x5A
0xF9 0x9F 0x84 0x1D 0xBA 0x19 0x3A 0xAA 0x11 0xA5
0xA1 0x09 0x72 0x02 0x85 0x9A 0x58 0xA2 0xDA 0x54
0xED 0x2A 0x57 0xF3 0x7F 0x00 0xBD 0xB0 0x31 0x0B
0x75 0xD5 0xCA 0xD0 0x3A 0x09 0x34 0x30 0x51 0xB9
0xF1 0x74 0x80 0xF8 0x79 0x8A 0x10 0xA1 0x16 0x89
0x40 0xCF 0xFC 0xDD 0x11 0x68 0xC2 0x22 0xF6 0xB5
0xFB 0xA3 0xED 0x44 0x81 0x1B 0xDA 0xBC 0xB4 0x2E
0xE3 0x52 0xA4 0x49 0x21 0xFD 0x9C 0x9F 0x41 0xF0
0xB7 0xD8 0x77 0xB4 0x62 0x3D 0x79 0x61 0x69 0xE9
0xD7 0x0A 0xA7 0x06 0x4C 0xD8 0x14 0xD8 0x9C 0xF1
0x56 0x1A 0xA9 0x42 0x06 0xD2 0x6C 0x70 0x28 0x04
0xE3 0xF4 0x11 0x14 0xC4 0x30 0x31 0x72]