Analyze the Security of the AEAD Scheme HiAE by Algebraic Techniques

Xichao Hu

State Key Laboratory of Cryptology, Beijing, China
xchao_h@163.com

Abstract. `HiAE` is a well-designed AEAD scheme using `AES` round functions, delivering outstanding performance on both ARM and x86 processors. Some existing AEAD schemes such as `Rocca` (ToSC 2021) have been shown to be vulnerable when attackers can repeatedly query the decryption oracle with forged ciphertexts and random tags until a valid tag is accepted. Motivated by this, we use algebraic techniques to analyze the security of `HiAE` under the same setting. Firstly, we employ the meet-in-the-middle technique and guess-and-determine technique to recover the state and derive a key-related equation resulting from two layers of `AES` round functions. Secondly, by adopting an algebraic approach to study the properties of the round function, we decompose the equation into byte-level equations for divide-and-conquer. Finally, we utilize the guess-and-determine technique to recover the key. Collectively, these techniques enable us to present the full key-recovery attack on `HiAE`. Our attack achieves a data complexity of 2^{130} and a time complexity of approximately 2^{209}, leveraging both encryption and decryption oracles with a success probability of 1. We emphasize that our attack considers a stronger scenario than `HiAE`'s original security model, and thus does not invalidate its original security claims.

Keywords: HiAE · Algebraic attack · Meet-in-the-middle · Guess-and-determine · Divide-and-conquer

1 Introduction

2 Introduction

An authenticated encryption with associated data (AEAD) scheme is a symmetric-key cryptographic primitive designed to provide authentication for associated data alongside encryption and authentication for messages. Among the multitude of proposed AEAD constructions, notable examples include `AEGIS` [WP13], `Tiaoxin` [Nik15], `Jean` and `Nikolić` [JN16], `Rocca` [SLN+21, SLN+22], and `HiAE` [CHPW25a]. These schemes all utilize the AES round function, combined with XOR operations, to achieve encryption and authentication for messages. This architectural choice is deliberate, as it maximizes utilization of hardware-accelerated instructions like AES-NI and SIMD, thereby enhancing computational efficiency. `HiAE` is meticulously engineered to deliver both high-performance and cross-platform efficiency across ARM and x86 architectures, addressing the stringent demands of next-generation communication systems like 6G networks and GPU/NPU interconnections. Through its innovative design, `HiAE` achieves remarkable efficiency: software implementation benchmarks across diverse platforms demonstrate throughput exceeding 340 Gbps on x86 processors and 180 Gbps on ARM devices in AEAD mode. This places `HiAE` as the fastest AEAD scheme on ARM chips to date while establishing a new performance milestone on the latest x86 processors. Thus, careful consideration

should be given to HiAE. It is necessary to verify the validity of the designers' statement: HiAE provides 256-bit security against state-recovery and key-recovery attacks in the nonce-respecting setting.

Our Contribution. In this paper, we investigate the security of HiAE under the setting where an adversary can repeatedly query the decryption oracle with forged ciphertexts and random tags until a valid tag is accepted. Using algebraic techniques, we present both state-recovery and key-recovery attacks on this scheme. Specifically,

- For the state-recovery attack, we first employ the meet-in-the-middle technique to build the relational expressions composed of AES round functions for the internal states, by analyzing the differential properties of those relational expressions, we ultimately succeeded in recovering the state with a probability of 1. The data complexity is 2^{130}, and the time complexity is 1 encryption and 2^{130} decryptions.

- For the key-recovery attack, based on the recovered state, we first apply the meet-in-the-middle and guess-and-determine techniques to derive a key-related equation as

$$\mathcal{A}(K_0 \oplus U_0) \oplus U_1 = \mathcal{A}^{-1}\left(\mathcal{A}(K_0 \oplus U_2) \oplus U_3\right) \oplus \mathcal{A}^{-1}(K_0 \oplus U_9) \oplus U_{17}.$$

  Here, $K_0, K_1 \in \mathbb{F}_2^{128}$ denote the keys of HiAE, $\mathcal{A}$ represents an AES round function, and $U_i (i \in \{0, 1, 2, 3, 9, 17\})$ are 128-bit constants. By leveraging an algebraic approach to analyze the properties of the round function, we decompose the equation into byte-level equations for divide-and-conquer. Finally, we successfully recover the key by reapplying the guess-and-determine technique[1]. The data complexity is 2^{130}, and the time complexity is 2^{209}.

Note that our attack on HiAE applies to the setting where the adversary can repeatedly query the decryption oracle with forged ciphertexts and random tags until a valid tag is accepted, which forms the basis of the attack in this paper. This scenario is not covered by the original security model of HiAE, and the rationale is discussed in Section 4.5 of the original paper [CHPW25a]. Nevertheless, we argue that our attack remains of significant importance for the design of AEAD schemes. The attack scenario considered in this paper was first applied to AEGIS [WP13], later extended to Rocca [HII+22, TTI24], and has gradually evolved into a general attack framework. Given that HiAE features an excellent design, superior performance, and sophisticated architecture, a thorough understanding of its design principles will greatly benefit the design and analysis of future AEAD schemes.
Organization. In Sect. 2, we present the necessary notation and provide a brief description for AES and HiAE. Sect. 3 presents our attack on HiAE. We conclude the paper in Sect. 4.

3 Preliminaries

3.1 Notations

- $\mathbb{S}$: The S-box of AES.

- $Q, \overline{Q}$: The MDS matrix and its inverse of AES.

- $\mathcal{A}$: $\mathcal{A}(x) = MC \circ SB \circ SR(x)$[2], where MC, SB and SR represent the operations $MixColumns, SubBytes, ShiftRows$ of the block cipher AES.

[1] In [BT25], Bille and Tischhauser independently proposed a key-recovery attack on HiAE. This work was carried out concurrently with ours, as shown by the similar ePrint timestamps of the initial versions of our paper [HJF+25] and [BT25]. Both papers have attracted the attention of the designers of HiAE, who provided a detailed note comparing the two works [CHPW25b].

[2] This is indeed the AES round function, but for efficiency purposes, it is represented in this way in [CHPW25a], and we follow this notation here.

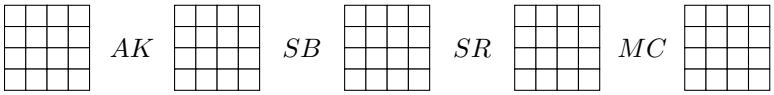


Figure 1: One round of block cipher `AES`

- S^i: The state in `HiAE` at i-th round, it consists of 16 blocks arranged as $S^i = (S^i_0, S^i_1, \ldots, S^i_{15})$, where each $S^i_j (0 \leq j \leq 15)$ represents a block, and S^i_0 is the first block.

- N: The 128-bit nonce.

- AD_i: The i-th 128-bit associated data block.

- M_i: The i-th 128-bit message block.

- C_i: The i-th 128-bit ciphertext.

- const_0, const_1: The 128-bit constant.

- $X \| 0^*$: a 128-bit string of the concatenation of X and the complement of zeros.

- $|M|$ or $len(M)$: The length of the string M.

3.2 Description of AES

One of the most well-known block ciphers globally is `AES` [DR02]. Its design concept has made a significant impression on the field of block ciphers. As a 128-bit block cipher, `AES` allows for key lengths of 128, 192, and 256 bits, with an 8-bit S-box. It belongs to the category of SPN ciphers and utilizes the MDS matrix to attain remarkable diffusion. The internal state can be seen as a square matrix of bytes presented below, in which each s_i is an element of $\mathbb{F}_2^8$ $(0 \leq i \leq 15)$.

$$S = \begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix}.$$

The encryption process of `AES` for one round, as shown in Figure 1, comprises the subsequent four operations:

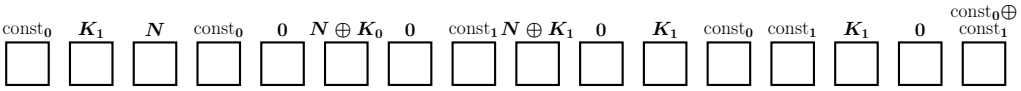
AddRoundKey (AK): The round key generated from the key schedule is XORed with the current state.

SubBytes (SB): The 8-bit S-box is applied in parallel to each byte of the cipher's internal state.

ShiftRows (SR): For the internal state matrix, the i-th row $(0 \leq i \leq 3)$ is circularly shifted to the left by i bytes.

MixColumns (MC): Each column of the internal state is multiplied by the MDS matrix Q, Q and its inverse matrix $\overline{Q}$ are detailed as follows.

$$Q = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}, \quad \overline{Q} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix}.$$



3.3 Description of HiAE

`HiAE` is the fastest AEAD solution on ARM chips to date while establishing a new performance milestone on the latest x86 processors. It takes the 256-bit key $K = K_0 \| K_1 (K_0, K_1 \in \mathbb{F}_2^{128})$, the 128-bit nonce $N \in \mathbb{F}_2^{128}$, the associated data AD, and the plaintext message M as the input, and output the ciphertext C whose length is the same as M and the 128-bit authentication tag T.

The processing of HiAE is divided into four phases: **initialization**, **associated data processing**, **encryption** and **finalization**. Note that before processing, both the associated data AD and the message M are zero-padded to ensure their lengths are multiples of 128 bits.

Initialization. In this phase, the key $K = K_0 \| K_1$ and nonce N are first loaded into the state S^{-32} as:

$$S_j^{-32} \leftarrow \begin{cases} \text{const}_0, \text{ if } j = 0, 3, 11, \\ K_1, \text{ if } j = 1, 10, 13, \\ N, \text{ if } j = 2, \\ N \oplus K_0, \text{ if } j = 5, \\ \text{const}_1, \text{ if } j = 7, 12, \\ N \oplus K_1, \text{ if } j = 8, \\ \text{const}_0 \oplus \text{const}_1, \text{ if } j = 15, \\ 0, \text{ otherwise}, \end{cases}$$

where $0 \le j \le 15$.

Then, the state undergoes 32 rounds of $\mathcal{F}_u$ with const_0 as input, where $\mathcal{F}_u$ is the update function of `HiAE`, as show in the black part of Figure 2, it products a new state $S^{i+1} = \mathcal{F}_u(S^i, X)$ according to the following process:

$$S_j^{i+1} = \begin{cases} \mathcal{A}(S_0^i \oplus S_1^i) \oplus \mathcal{A}(S_{13}^i) \oplus X, \text{ if } j = 15, \\ S_{j+1}^i \oplus X, \text{ if } j = 2, 12, \\ S_{j+1}^i, \text{ otherwise}, \end{cases}$$

where $0 \le j \le 15$, and $X = \text{const}_0$ in this phase. Finally, the keys K_0 and K_1 are XORed into S_9^0 and S_{13}^0, respectively.

Processing the associated data. Each 128-bit block of associated data AD_i updates the state through the $\mathcal{F}_u$: $S^{i+1} = \mathcal{F}_u(S^i, AD_i)(0 \le i \le |AD|/128 - 1)$. This phase is skipped if the associated data is empty.

Encryption. As shown in the blue part of Figure 2 ($X = M_i$), for $i = 0$ to $|M|/128 - 1$, the i-th block of message M_i is encrypted to produce C_i:

$$C_i = \mathcal{A}(S_0^i \oplus S_1^i) \oplus S_9^i \oplus M_i.$$

The last block of ciphertext is then truncated so that the length of the ciphertext is equal to the length of the message. The encryption phase is skipped if the message is empty.

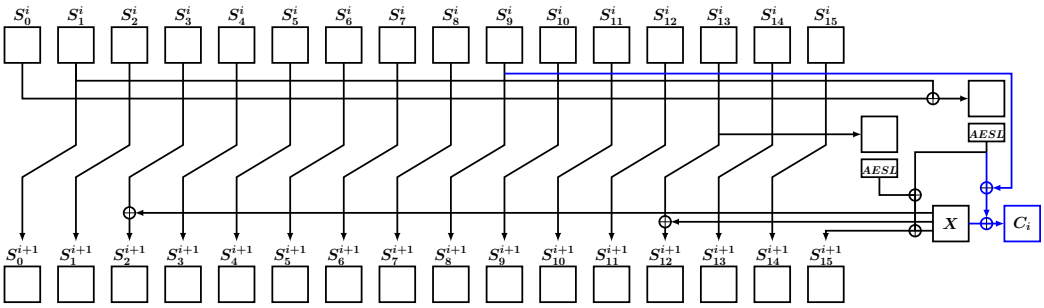


Figure 2: The update function and one round encryption of `HiAE`

Finalization. The lengths of the associated data and the message are used to update the state as:

$$S^{i+1} = \mathcal{F}_u(S^i, len(AD)||len(M))$$

and the tag is computed as:

$$T = \bigoplus_{j=0}^{15} S_j^r,$$

where r is the number of rounds for finalization.

4 Key-recovery Attack on HiAE

An authenticated encryption with associated data (AEAD) scheme is a symmetric-key cryptographic primitive designed to provide authentication for associated data alongside encryption and authentication for messages. `HiAE` is a nonce-based AEAD scheme, it outputs a 128-bit tag and claims 256-bit security against key-recovery attacks. As a nonce-based AEAD scheme, the attackers are prohibited from submitting different messages with the same nonce to the encryption oracle, which constitutes a very strong constraint. However, when an attacker can repeatedly query the decryption oracle with forged ciphertexts and random tags until a valid tag is accepted, even if the attacker follows the nonce-respecting rule for the encryption oracle, the attack can still obtain nonce-repeated pairs as described below.

1. The attacker queries the encryption oracle (N, M) to get (C, T), where N denotes an arbitrary nonce and M represents a message with a strategically chosen length.

2. The attacker introduces a carefully crafted difference, denoted as Δ, into the ciphertext. Subsequently, they initiate a decryption query using the tuple $(N, C \oplus \Delta, T')$, where they systematically test q_d potential values for T'. This process yields a "nonce-repeated" plaintext (N, M') with a probability of $\frac{q_d}{2^{128}}$.

The above process represents a sophisticated method for collecting nonce-repeated pairs. Based on this approach, in ToSC 2022/FSE 2023, Hosoyamada et al. showed vulnerabilities in the AEAD scheme `Rocca` (ToSC 2021) in the nonce-respecting setting [HII+22]. In detail, they make one encryption query and 2^{128} decryption queries, and recover the entire 256-bit key with probability almost 1, where the time complexity is about 2^{128}. As `Rocca` has a 256-bit key, this breaks the designers' security claim regarding key recovery. Hosoyamada et al. also suggested countermeasures, and one of them is to XOR the key at the end of the initialization and at the beginning of the finalization.

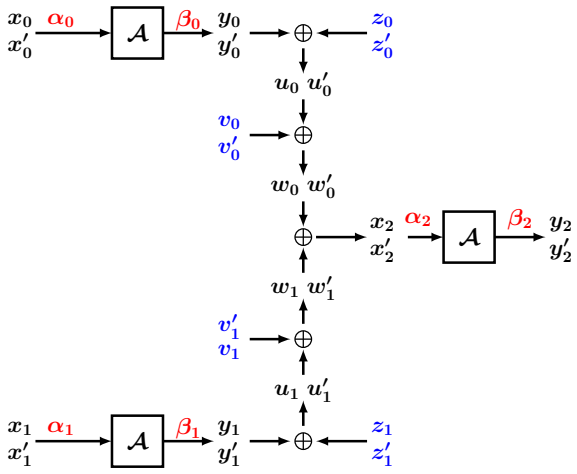


Figure 3: Illustration of a Property of $\mathcal{A}$

In reaction to the attack proposed by Hosoyamada et al., the designers of `Rocca` first revised the version by incorporating the key XOR operation at the end of the initialization and the start of the finalization. However, in the subsequent ePrint version [SLN+22], the key XOR operation at the beginning of the finalization stage was removed, which has sparked worries about potential security risks. In ToSC 2024, Takeuchi et al. conducted a thorough security analysis of all above versions of `Rocca` [TTI24], the version of `Rocca` which integrates a key XOR operation at the end of the initialization remains secure against key-recovery attacks.

In the following, we analyze the security of `HiAE` under the setting beyond that considered in its original paper [CHPW25a], where an attacker can repeatedly query the decryption oracle with forged ciphertexts and random tags until a valid tag is accepted. We first propose a state-recovery attack, and then put forward a key-recovery attack.

4.1 The state-recovery attack on HiAE

First of all, we elaborate on a commonly-known feature of the input-output differences in the `AES` S-box $\mathbb{S}$. It serves as an alternative interpretation of Observation 1 mentioned 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 $\mathbb{S}(x) \oplus \mathbb{S}(x \oplus \Delta_{in}) = \Delta_{out}$. 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 $1/32$.*

From Lemma 1, given an input-output difference of the `AES` S-box, the number of candidates of corresponding input-output values is at most 2^2. This property allows us to derive the following theorem about the property of $\mathcal{A}$.

Theorem 1. *Let $(\alpha_0, \beta_0), (\alpha_1, \beta_1), (\alpha_2, \beta_2) \in (\mathbb{F}_2^{128})^2$ be three known input-output differences, and let $z_0, z'_0, z_1, z'_1, u_0, u'_0, u_1, u'_1 \in \mathbb{F}_2^{128}$ be eight known arbitrary values. If α_0 and α_1 activate all S-boxes in $\mathcal{A}$, then there exists a unique solution $(x_0, x'_0, x_1, x'_1, y_2, y'_2) \in$*

$(\mathbb{F}_2^{128})^6$ *to the following equations:*

$$\begin{cases} y_0 = \mathcal{A}(x_i), y_0' = \mathcal{A}(x_i') \quad (0 \le i \le 2), \\ \alpha_i = x_i \oplus x_i', \beta_i = y_i \oplus y_i' \quad (0 \le i \le 2), \\ u_i = y_i \oplus z_i, u_i' = y_i' \oplus z_i' \quad (0 \le i \le 1), \\ w_i = u_i \oplus v_i, w_i' = u_i' \oplus v_i' \quad (0 \le i \le 1), \\ x_2 = w_0 \oplus w_1, x_2' = w_0' \oplus w_1', \end{cases} \tag{1}$$

where $y_0, y_0', u_0, u_0', w_0, w_0', y_1, y_1', u_1, u_1', w_1, w_1', x_2, x_2' \in \mathbb{F}_2^{128}$ *are auxiliary variables.*

Proof. The entire proof process is shown in Figure 3. On the one hand, since α_0 activates all S-boxes in $\mathcal{A}$, according to Lemma 1, for fixed (α_0, β_0), there are at most $(2^2)^{16} = 2^{32}$ pairs (x_0, x_0') such that $x_0 \oplus x_0' = \alpha_0$ and $y_0 \oplus y_0' = \mathcal{A}(x_0) \oplus \mathcal{A}(x_0') = \beta_0$. Analogously, there are at most 2^{32} pairs (x_1, x_1') such that $x_1 \oplus x_1' = \alpha_1$ and $y_1 \oplus y_1' = \mathcal{A}(x_1) \oplus \mathcal{A}(x_1') = \beta_1$. Consequently, the total number of pairs (x_2, x_2') is at most 2^{64}. On the other hand, for fixed differences $\alpha, \beta \in \mathbb{F}_2^{128}$ and an arbitrary value $x \in \mathbb{F}_2^{128}$, the probability that $\mathcal{A}(x) \oplus \mathcal{A}(x \oplus \alpha) = \beta$ is $1/2^{128}$. Therefore, the probability that an incorrect value of $(x_0, x_0', x_1, x_1', y_2, y_2')$ survives is at most $2^{64}/2^{128} < 1$. As the correct value of $(x_0, x_0', x_1, x_1', y_2, y_2')$ must satisfy Equation (1), the proof is complete. □

Next, by using the property of $\mathcal{A}$, we apply the meet-in-the-middle technique to show that we can recover the internal state by injecting a difference into the ciphertext.

Theorem 2. *Let h be a integer, N be an arbitrary nonce, M, M' be two arbitrary messages each of length at least $14 + h$ blocks, C, C' be two arbitrary ciphertexts, and T be a tag. If (C, T) is the encryption result of (N, M) and (C', T) is the encryption result of (N, M'), and $C' = C \oplus \Delta$, where Δ is a difference injected into C_h and it activates all S-boxes in $\mathcal{A}$, then we can recover the internal state $(S_0^{h+11}, S_1^{h+11}, S_2^{h+11}, S_6^{h+11}, S_9^{h+11}, S_{10}^{h+11})$.*

Proof. We only prove the case when $h = 0$, and other cases can be proved similarly. By injecting the difference Δ into C_0, we obtain the differential transitions in the decryption process as shown in Figure 4 and Figure 5, where the white color denotes that the differences before SB are constant, the yellow color denotes that the differences before SB are known and non-zero, the blue color denotes that the differences before SB are known, the red color denotes that the differences before SB are non-zero, and the gray color denotes that the differences before SB are unknown. The notations A_e^r and A_u^r represent the function $\mathcal{A}$, and the position label with X also corresponds to the positions of X' and ΔX. The red-colored number shows the order of the 128-bit state whose candidates are narrowed down following the step number.

Since the input and output differences of A_e^2 and A_e^3 are known and activate all S-boxes, the values M_2, M_2', C_2, C_2', M_3, M_3', C_3, C_3', M_8, M_8', and M_9, M_9' are all known. Additionally, since the input and output differences of A_e^{11} are known, we can recover the state $(S_0^{11}, S_1^{11}, S_9^{11})$ according to Theorem 1. Analogously, we can recover the state $(S_0^{12}, S_1^{12}, S_9^{12})$. Furthermore, given that S_{15}^5, S_9^4, C_4, and M_4 are known, S_{13}^4 can be determined. Altogether, we recover the state $(S_0^{11}, S_1^{11}, S_2^{11}, S_6^{11}, S_9^{11}, S_{10}^{11})$. □

Finally, we adopt the following process to recover the entire state under the setting where an attacker can repeatedly query the decryption oracle with forged ciphertexts and random tags until a valid tag is accepted.

1: Query the encryption oracle (N, M) to obtain (C, T), where N denotes an arbitrary nonce and M represents arbitrary messages whose length is at least 23 blocks.

2: For $r \in \{0, 3, 5, 7\}$, repeat the following steps:

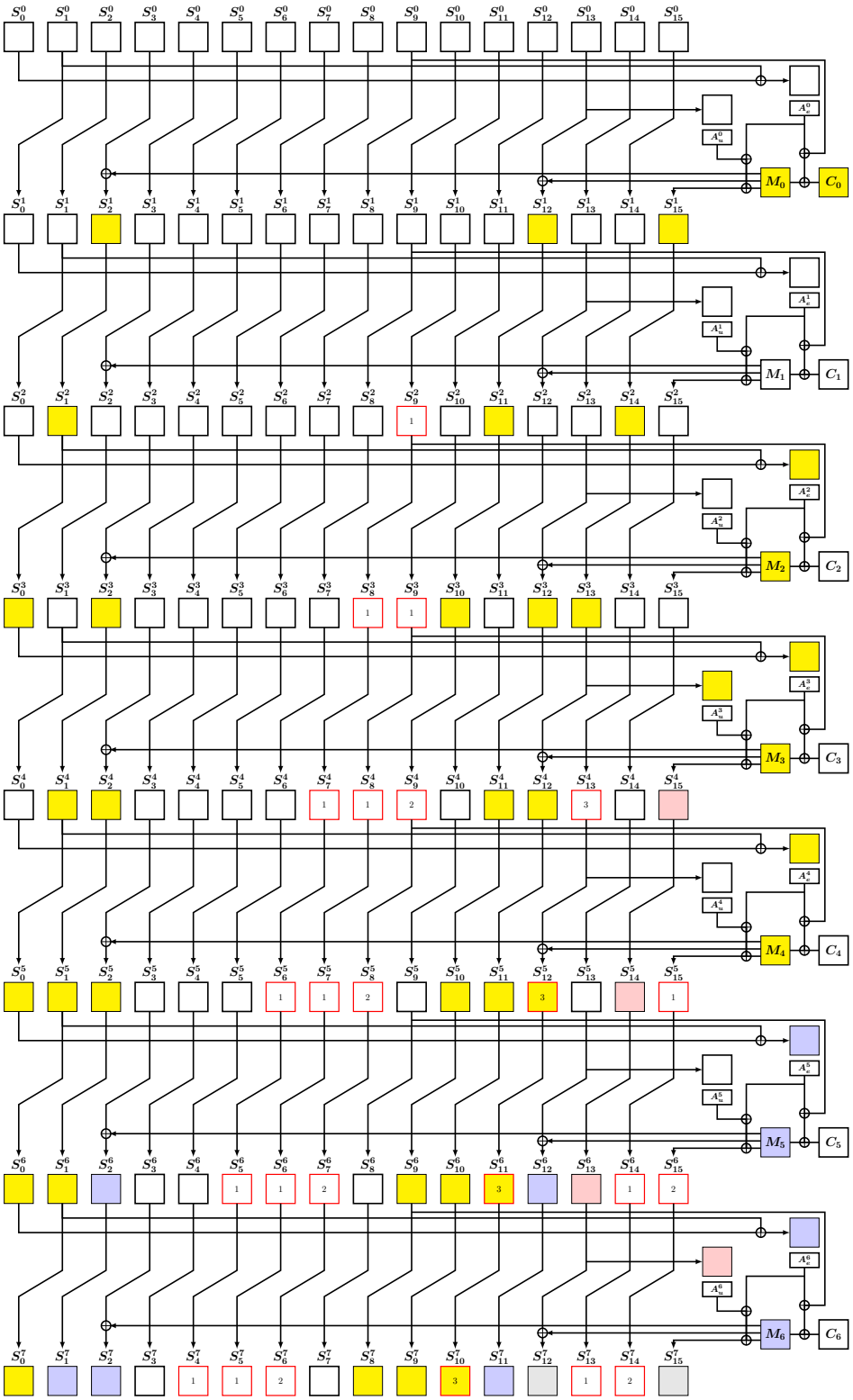


Figure 4: The round 0-6 of the state recovery attack against `HiAE`

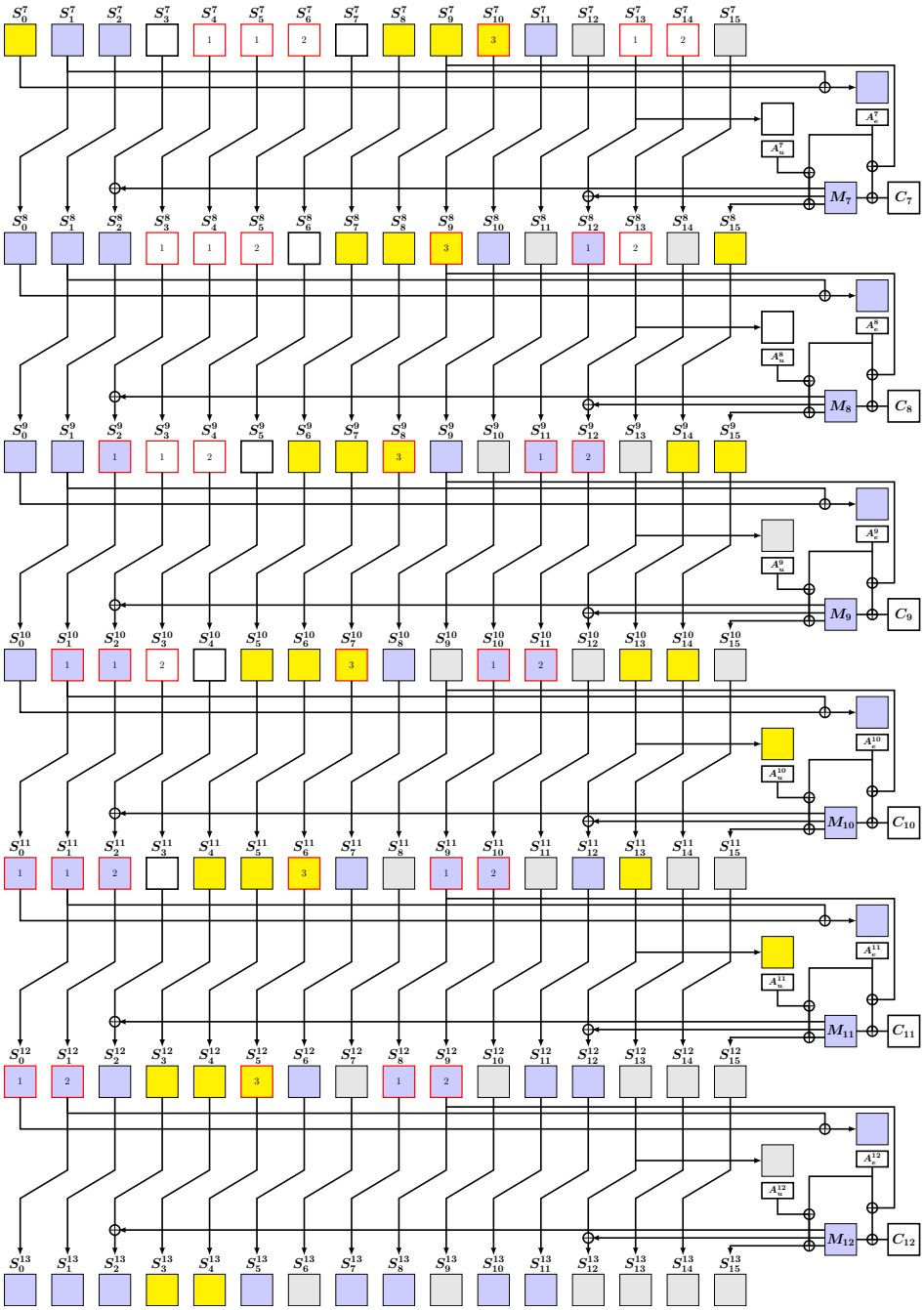


Figure 5: The round 7-13 of the state recovery attack against `HiAE`

2.1: Inject the difference Δ into C_i, where Δ activates all S-boxes in $\mathcal{A}$.

2.2: Initiate a decryption query using the tuple $(N, C \oplus \Delta, T')$, and systematically test q_d potential values for T'. This yields a nonce-repeated plaintext (N, M') such that its encryption result is $(C \oplus \Delta, T)$.

2.3: Recover the states $(S_0^{r+11}, S_1^{r+11}, S_2^{r+11}, S_6^{r+11}, S_9^{r+11}, S_{10}^{r+11})$ according to Theorem 2.

The values of $(S_0^{r+11}, S_1^{r+11}, S_2^{r+11}, S_6^{r+11}, S_9^{r+11}, S_{10}^{r+11})$ for $r \in \{0, 3, 5, 7\}$ can be used to recover the entire state of S^{11}, which is equivalent to recovering S^0.

Complexity. In Step 1, we require one encryption. In Step 2.2, since the tag size of `HiAE` is 128 bits, the data complexity and time complexity of this step both are 2^{128}. In Step 2.3, the time complexity is no more than 2^{64} one round encryption of `AES`, which is negligible compared to Step 2.2. Thus, for the state-recovery attack under our setting, the data complexity is $4 \times 2^{128} = 2^{130}$, the time complexity is 1 encryption and $4 \times 2^{128} = 2^{130}$ decryptions.

4.2 The key-recovery attack against HiAE

In this part, on the basis of the already recovery the state, we present a key-recovery attack against `HiAE` under the setting where an adversary can repeatedly query the decryption oracle with forged ciphertexts and random tags until a valid tag is accepted. First, we employ the meet-in-the-middle technique and guess-and-determine technique to derive a key-related equation. In details, we guess the key K_1, and propagate K_0 from round -32 to round -11 in the forward direction and from round 0 to round -11 in the backward direction. Then all states influenced solely by K_1 can be regarded as a constant. As shown in Figures 6 and 7, we derive the key-related equation as follows, where $U_i (0 \leq i \leq 17)$ are constants.

- In the forward direction:

1. Since $S_5^{-32} = K_0 \oplus N$ and $S_0^{-27} = S_5^{-32}$, we have $S_0^{-27} = K_0 \oplus N$.

2. Since $S_0^{-27} = K_0 \oplus N$, S_1^{-27} and S_{13}^{-27} are constant, let $U_0 = N \oplus S_1^{-27}$ and $U_1 = \mathcal{A}(S_{13}^{-27}) \oplus \text{const}_0$. Then U_0 and U_1 are constants, and $S_{15}^{-26} = \mathcal{A}(K_0 \oplus U_0) \oplus U_1$.

3. Since $S_0^{-11} = S_1^{-12} = S_{15}^{-26}$, it follows that $S_0^{-11} = \mathcal{A}(K_0 \oplus U_0) \oplus U_1$.

- In the backward direction:

1. As the end of the initialization, two 128-bit keys are XORed with the state: $S_9^0 = S_9^0 \oplus K_0$, $S_{13}^0 = S_{13}^0 \oplus K_1$, and the state after the initialization is already recovered. Then, the state S_9^0 at the initialization can be represent as $S_9^0 = K_0 \oplus U_2$, where U_2 is a known constant. Simultaneously, since K_1 has been guessed, S_{13}^0 at the initialization can be regarded as a constant value.

2. Since $S_{12}^{-3} = S_9^0$, we have $S_{12}^{-3} = K_0 \oplus U_2$.

3. Since S_{15}^{-3} and S_1^{-4} are constants, let $U_3 = S_{15}^{-3} \oplus \text{const}_0$ and $U_4 = S_1^{-4}$. Then, $S_0^{-4} = \mathcal{A}^{-1}\left(\mathcal{A}(K_0 \oplus U_2) \oplus U_3\right) \oplus U_4$.

4. Since S_{15}^{-4} and S_{13}^{-5} are constants, let $U_5 = \mathcal{A}^{-1}\left(\mathcal{A}(S_{13}^{-5}) \oplus S_{15}^{-4} \oplus \text{const}_0\right)$ and $U_6 = U_4 \oplus U_5$. Then, $S_0^{-5} = \mathcal{A}^{-1}\left(\mathcal{A}(K_0 \oplus U_2) \oplus U_3\right) \oplus U_6$.

5. Since S_{15}^{-5} and S_{13}^{-6} are constants, let $U_7 = \mathcal{A}^{-1}\left(\mathcal{A}(S_{13}^{-6}) \oplus S_{15}^{-5} \oplus \text{const}_0\right)$ and $U_8 = U_6 \oplus U_7$. Then, $S_0^{-6} = \mathcal{A}^{-1}\left(\mathcal{A}(K_0 \oplus U_2) \oplus U_3\right) \oplus U_8$.

6. Since $S_{15}^{-6} = K_0 \oplus U_2$ and S_{13}^{-7} is a constant, let $U_9 = \mathcal{A}(S_{13}^{-7}) \oplus U_2 \oplus \text{const}_0$. Then $S_0^{-7} = \mathcal{A}^{-1}\left(\mathcal{A}(K_0 \oplus U_2) \oplus U_3\right) \oplus \mathcal{A}^{-1}(K_0 \oplus U_9) \oplus U_8$.

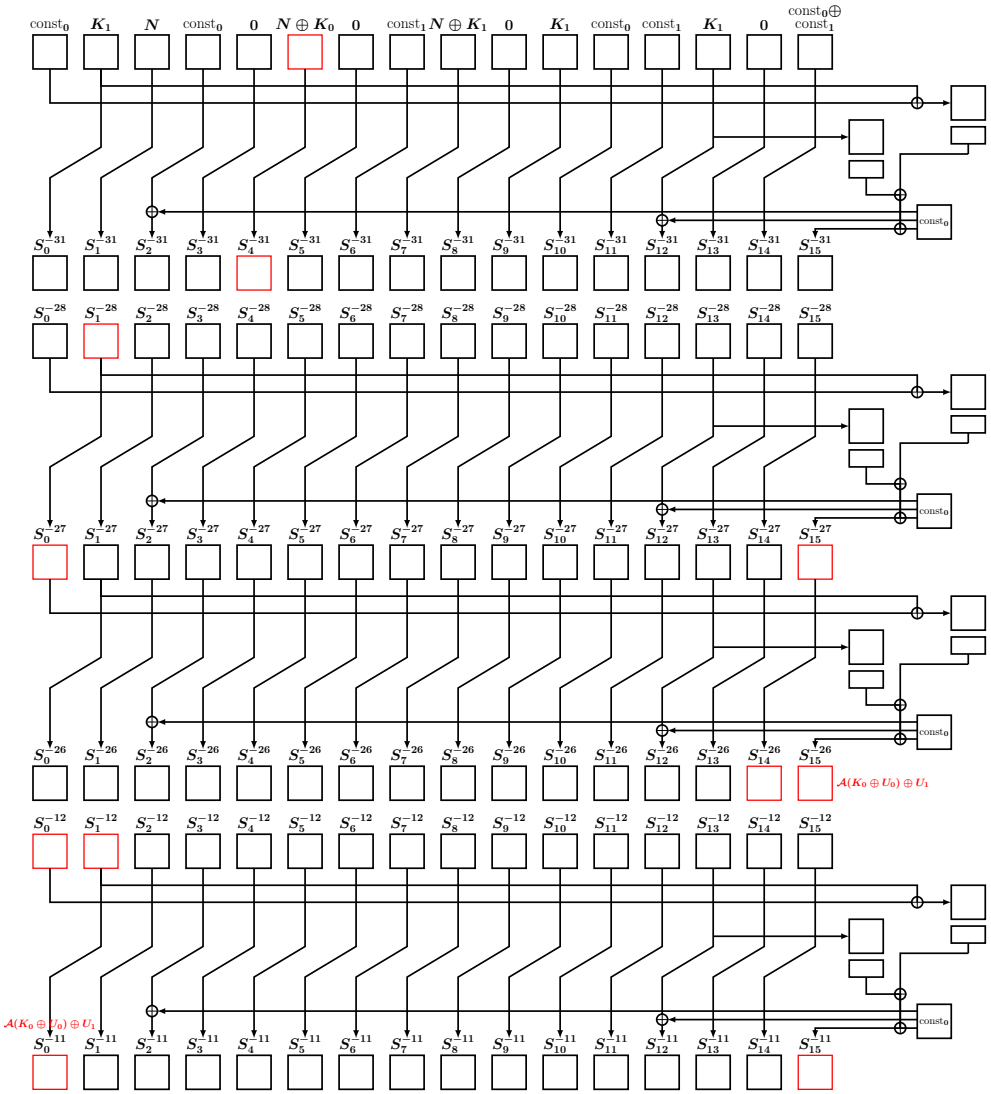


Figure 6: Propagate K_0 from round -32 to round -11 in the forward direction

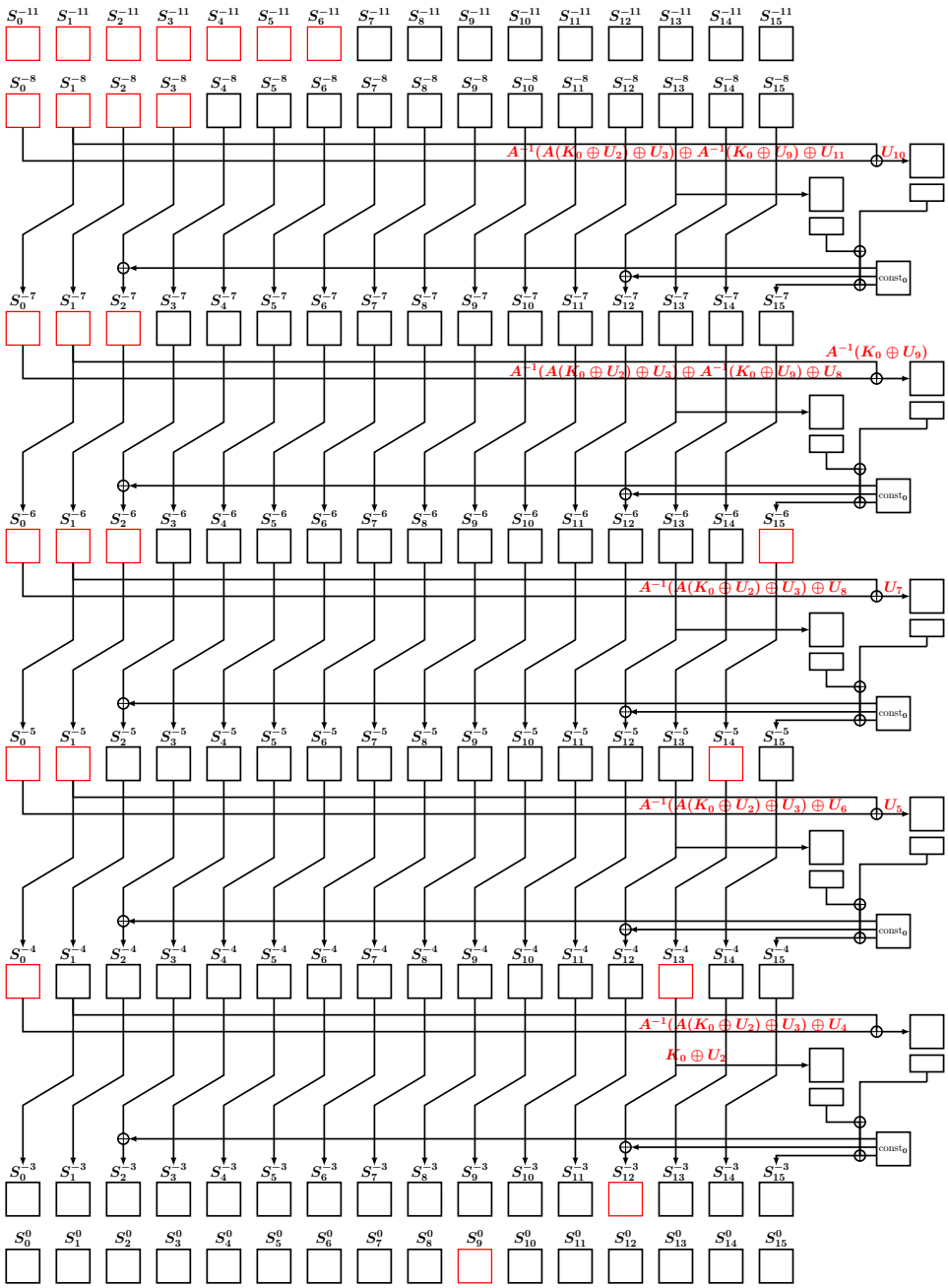


Figure 7: Propagate K_0 from round 0 to round -11 in the backward direction

7. Since $S_{15}^{-(r-1)}$ and S_{13}^{-r} are constants, let $U_{2r-6} = \mathcal{A}(S_{13}^{-r}) \oplus S_{15}^{-(r-1)} \oplus \text{const}_0$ and $U_{2r-5} = U_{2r-6} \oplus U_{2r-7}$. Then, $S_0^{-r} = \mathcal{A}^{-1}\left(\mathcal{A}(K_0 \oplus U_2) \oplus U_3\right) \oplus \mathcal{A}^{-1}(K_0 \oplus U_9) \oplus U_{2r-5}$ for $r = 8, \ldots, 11$.

Thus, we have

$$\mathcal{A}(K_0 \oplus U_0) \oplus U_1 = S_0^{-11} = \mathcal{A}^{-1}\left(\mathcal{A}(K_0 \oplus U_2) \oplus U_3\right) \oplus \mathcal{A}^{-1}(K_0 \oplus U_9) \oplus U_{17}. \quad (2)$$

Next, we conduct a detailed study of the round function of `AES` and decompose Equation (2) into byte-level equations regarding the keys. Let $T^0 = \mathcal{A}(K_0 \oplus U_0)$, $T^1 = \mathcal{A}^{-1}(K_0 \oplus U_9)$, and $T^2 = \mathcal{A}^{-1}\left(\mathcal{A}(K_0 \oplus U_2) \oplus U_3\right)$, then

$$T^0 \oplus T^1 \oplus T^2 = U_1 \oplus U_{17}. \quad (3)$$

Let $T^3 = MC^{-1}(K_0 \oplus U_9)$ and $T^4 = MC^{-1}(U_3)$, then T^4 is a constant and

$$\begin{aligned} T^2 &= \mathcal{A}^{-1}\left(\mathcal{A}(K_0 \oplus U_2) \oplus U_3\right) \\ &= (SR^{-1} \circ SB^{-1} \circ MC^{-1})(MC \circ SB \circ SR(K_0 \oplus U_2) \oplus U_3), \\ &= SR^{-1} \circ SB^{-1}(SB \circ SR(K_0 \oplus U_2) \oplus MC^{-1}U_3), \\ &= SR^{-1} \circ SB^{-1}(SB \circ SR(K_0 \oplus U_2) \oplus T^4), \\ &= SB^{-1}(SB(K_0 \oplus U_2) \oplus SR^{-1}(T^4)). \end{aligned}$$

Thus, for $0 \le t \le 15$, let t_0 and t_1 be the integer quotient when t is divided by 4, we have the following byte-level equations, where $K_{0,t}, U_{0,t}, U_{9,t}, T_t^0, T_t^1, T_t^2, T_t^3$ are the t-th bytes of $K_0, U_0, U_9, T^0, T^1, T^2, T^3$ respectively.

$$\begin{cases} T_t^0 = \bigoplus_{i=0}^{3} Q_{t_0,i} \cdot \mathbb{S}(K_{0,4((t_1+i) \mod 4)+i} \oplus U_{0,4((t_1+i) \mod 4)+i}), \\ T^3_{4((t_1-t_0) \mod 4)+t_0} = \bigoplus_{i=0}^{3} \overline{Q}_{t_0,i} \cdot (K_{0,4((t_1-t_0) \mod 4)+i} \oplus U_{9,4((t_1-t_0) \mod 4)+i}), \\ T_t^1 = \mathbb{S}^{-1}(T^3_{4((t_1-t_0) \mod 4)+t_0}), \\ T_t^2 = \mathbb{S}^{-1}(\mathbb{S}(K_{0,t} \oplus U_{2,t}) \oplus T^4_{4((t_1-t_0) \mod 4)+t_0}). \end{cases}$$

Since $T_t^0 \oplus T_t^1 \oplus T_t^2$ is known from Equation (3), we can derive 16 byte-level equations, the bytes of key K_0 that contained in each byte-level equation is shown in Table 1. Finally, with these byte-level equations, we apply the guess-and-determine technique to recover the key K_0 as follows:

1. Guess $K_{0,0}, K_{0,1}, K_{0,2}, K_{0,3}, K_{0,5}, K_{0,10}, K_{0,15}$, and check $T_t^0 \oplus T_t^1 \oplus T_t^2 = U_{1,t} \oplus U_{17,t} (t = 0)$ to sieve the keys. The time complexity of this step is 2^{56}, and 2^{48} keys are remained.

2. Guess $K_{0,12}, K_{0,13}, K_{0,14}$, and check $T_t^0 \oplus T_t^1 \oplus T_t^2 = U_{1,t} \oplus U_{17,t} (t = 1)$ to sieve the keys. The time complexity of this step is $2^{48+24} = 2^{72}$, and 2^{64} keys are remained.

3. Guess $K_{0,4}, K_{0,9}$, and check $T_t^0 \oplus T_t^1 \oplus T_t^2 = U_{1,t} \oplus U_{17,t} (t = 5, 6)$ to sieve the keys. The time complexity of this step is $2^{64+16} = 2^{80}$, and 2^{64} keys are remained.

4. Guess $K_{0,8}, K_{0,11}$, and check $T_t^0 \oplus T_t^1 \oplus T_t^2 = U_{1,t} \oplus U_{17,t} (t = 2, 7)$ to sieve the keys. The time complexity of this step is $2^{64+16} = 2^{80}$, and 2^{64} keys are remained.

5. Guess $K_{0,7}$, and check $T_t^0 \oplus T_t^1 \oplus T_t^2 = U_{1,t} \oplus U_{17,t} (t = 8, 10, 11)$ to sieve the keys. The time complexity of this step is $2^{64+8} = 2^{72}$, and 2^{48} keys are remained.

6. Guess $K_{0,6}$, and check the remained bytes of $T_t^1 \oplus T_t^2 = U_{1,t} \oplus U_{17,t} (t = 3, 4, 9, 12, 13, 14, 15)$ to sieve the keys. The time complexity of this step is $2^{48+8} = 2^{56}$, and 1 keys are remained.

Table 1: The K_0 associated with $T_t^0 \oplus T_t^1 \oplus T_t^2 = U_{1,t} \oplus U_{17,t} (0 \le t \le 15)$

t	Keys	Guessed order	Time complexity
0	$K_{0,0}, K_{0,1}, K_{0,2}, K_{0,3}, K_{0,5}, K_{0,10}, K_{0,15}$	1	2^{56}
1	$K_{0,0}, K_{0,5}, K_{0,10}, K_{0,12}, K_{0,13}, K_{0,14}, K_{0,15}$	2	$2^{56-8} \cdot 2^{24}$
5	$K_{0,0}, K_{0,1}, K_{0,2}, K_{0,3}, K_{0,4}, K_{0,9}, K_{0,14}$	3	$2^{80-16} \cdot 2^{16}$
6	$K_{0,3}, K_{0,4}, K_{0,9}, K_{0,12}, K_{0,13}, K_{0,14}, K_{0,15}$		
2	$K_{0,0}, K_{0,5}, K_{0,8}, K_{0,9}, K_{0,10}, K_{0,11}, K_{0,15}$	4	$2^{96-32} \cdot 2^{16}$
7	$K_{0,3}, K_{0,4}, K_{0,8}, K_{0,9}, K_{0,10}, K_{0,11}, K_{0,14}$		
8	$K_{0,2}, K_{0,7}, K_{0,8}, K_{0,9}, K_{0,10}, K_{0,11}, K_{0,13}$	5	$2^{112-48} \cdot 2^{8}$
10	$K_{0,0}, K_{0,1}, K_{0,2}, K_{0,3}, K_{0,7}, K_{0,8}, K_{0,13}$		
11	$K_{0,2}, K_{0,7}, K_{0,8}, K_{0,12}, K_{0,13}, K_{0,14}, K_{0,15}$		
3	$K_{0,0}, K_{0,4}, K_{0,5}, K_{0,6}, K_{0,7}, K_{0,10}, K_{0,15}$	6	$2^{120-72} \cdot 2^{8}$
4	$K_{0,3}, K_{0,4}, K_{0,5}, K_{0,6}, K_{0,7}, K_{0,9}, K_{0,14}$		
9	$K_{0,2}, K_{0,4}, K_{0,5}, K_{0,6}, K_{0,7}, K_{0,8}, K_{0,13}$		
12	$K_{0,1}, K_{0,6}, K_{0,11}, K_{0,12}, K_{0,13}, K_{0,14}, K_{0,15}$		
13	$K_{0,1}, K_{0,6}, K_{0,8}, K_{0,9}, K_{0,10}, K_{0,11}, K_{0,12}$		
14	$K_{0,1}, K_{0,4}, K_{0,5}, K_{0,6}, K_{0,7}, K_{0,11}, K_{0,12}$		
15	$K_{0,0}, K_{0,1}, K_{0,2}, K_{0,3}, K_{0,6}, K_{0,11}, K_{0,12}$		

Complexity. After we recovery the state, the complexity of recovering K_0 and K_1 is $2^{128} \times 2 \times 2^{80} = 2^{209}$, and the data complexity is 1. All in all, for the entire key recovery process, the data complexity is $4 \times 2^{128} = 2^{130}$, and the time complexity is $2^{128} \times 2 \times 2^{80} = 2^{209}$. As $2^{209} < 2^{256}$, we recover the key under our attack setting.

5 Conclusion

In this paper, we employ algebraic techniques to evaluate the security of `HiAE` under the setting where an adversary can repeatedly query the decryption oracle with forged ciphertexts and random tags until a valid tag is accepted. As a result, we present a full key-recovery attack, which has a data complexity of 2^{130} and a time complexity of approximately 2^{209}. This attack leverages both encryption and decryption oracles, with a success probability of 1. Our attack setting lies outside the scope of the original security model of `HiAE`, and thus does not impact the security guarantees of `HiAE`. However, owing to `HiAE`'s excellent design, superior performance, and sophisticated architecture, research into its security will significantly advance the design of future AEAD schemes.

References

[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.

[BT25] Alexander Bille and Elmar Tischhauser. Cryptanalysis of HiAE. Cryptology ePrint Archive, Paper 2025/1180, 2025.

[CHPW25a] Han Chen, Tao Huang, Phuong Pham, and Shuang Wu. HiAE: A high-throughput authenticated encryption algorithm for cross-platform efficiency. Cryptology ePrint Archive, Paper 2025/377, 2025.

[CHPW25b] Han Chen, Tao Huang, Phuong Pham, and Shuang Wu. HiAE remains secure in its intended model: A clarification of claimed attacks. Cryptology ePrint Archive, Paper 2025/1235, 2025.

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

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

[HJF+25] Xichao Hu, Lin Jiao, Dengguo Feng, Yonglin Hao, Senpeng Wang, Yongqiang Li, and Xinxin Gong. Breaking the authenticated encryption scheme HiAE. Cryptology ePrint Archive, Paper 2025/1203, 2025.

[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.

[Nik15] Ivica Nikolić. Tiaoxin-346. 2015.

[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.

[TTI24] Ryunouchi Takeuchi, Yosuke Todo, and Tetsu Iwata. Key recovery, universal forgery, and committing attacks against revised rocca: How finalization affects security. *IACR Trans. Symmetric Cryptol.*, 2024(2):85–117, 2024.

[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.