Practical Evaluation and Complexity Analysis of Forgery Attacks on the PAES-8 Authenticated Encryption Scheme

Susila Windarta[1]*, Imas Purbasari[2]
[1]*Politeknik Siber dan Sandi Negara*
[1]*Jalan Raya Haji Usa, Putat Nutug, Ciseeng, Bogor 16120*
[2]*Badan Siber dan Sandi Negara*
[2]*Jalan Harsono RM No. 70, Ragunan, Pasar Minggu, Jakarta Selatan 12550*
Email: susila.windarta@poltekssn.ac.id*

ARTICLE INFORMATION
Received 30 August 2024
Revised 23 October 2024
Accepted 12 December 2024
Keywords:
AEAD
Authenticated encryption
Forgery attack
Nonce-misuse
PAES-8

A B S T R A C T
The Parallelizable Authenticated Encryption Scheme (PAES)-8, designed by Ye et al. in 2014, claims to provide 128-bit authentication security in the nonce-misuse model. However, Sasaki and Wang's theoretical forgery attack on PAES-8 exposed vulnerabilities, suggesting a universal forgery with a complexity of approximately 2^{11}. This study presents a practical implementation of Sasaki and Wang's theoretical forgery attack on the PAES-8 encryption scheme, uncovering significant modifications required for its execution. This including the use of DDT-based plaintext injection, staged state recovery, multiple injection attempts, and algorithmic adjustments. Our findings demonstrate that these modifications increase the attack complexity to approximately $2^{11} + 2^{12} + 2^{7} \approx 2^{12}$, indicating greater resistance in PAES-8 than previously anticipated. Future cryptanalysis should focus on exploring nonce-respecting models to evaluate the scheme's security.

1. Introduction

A recent report by the World Economic Forum (Charlton, 2024) highlights that the global cost of cybercrime is projected to reach $23.84 trillion by 2027, a significant increase from $8.44 trillion in 2022. This sharp rise underscores the growing severity and financial impact of cyberattacks, which have become a critical concern for global leaders and businesses. The report also notes that 2023 witnessed several major cyberattacks, including those targeting the US State Department, reflecting the increasing sophistication and frequency of these threats.

In response to these escalating cyber threats, security technologies such as Authenticated Encryption (AE) schemes have become essential for ensuring data confidentiality and authenticity (Jimale et al., 2022). By integrating encryption and authentication into a single operation, AE schemes offer robust protection against a wide array of attacks that threaten the security of digital communications. As cyber threats continue to evolve (Statista, 2024), the resilience of these schemes under various attack models becomes increasingly crucial.

The concept of AE schemes was introduced by Bellare and Namprempre in 2000 and further developed (Bellare & Namprempre, 2008). NIST and Daniel J. Bernstein organized a competition to identify an AE scheme, the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) (Bernstein, 2013). CAESAR ran for four rounds. One of the CAESAR candidates submitted in the first round was the Parallelizable Authenticated Encryption Scheme (PAES), designed by Ye et al. (2014). Other algorithms, such as Ascon (Dobraunig et al., 2021), Deoxys (Jean et al., 2021), and NORX (Aumasson et al., 2015) are noted for their high performance, while OCB3 (Bhaumik & Nandi, 2017), and AEGIS (Wu & Preneel, 2013) emphasize robust security. Lightweight designs like GIFT-COFB (Banik et al., 2021), Photon-Beetle (Bao et al., 2021), Saturnin (Canteaut et al., 2020), Xoodyak (Daemen et al., 2020), and ZLR (Choi et al., 2024) cater to resource-constrained environments, demonstrating the adaptability of modern cryptographic solutions.

PAES is an AE scheme built on the round function of the AES block cipher algorithm (NIST, 2001). PAES consists of two structures, PAES-4 and PAES-8. Both structures are similar except for the number of states used. PAES-4 uses four state blocks, while PAES-8 uses eight state blocks. Each state block is 128 bits, which will be

used in the encryption process along with 128 bits of the key cap K, 128 bits of the nonce cap N, 128 bits of associated data, and plaintext of arbitrary size.

There are two types of attacks on AE schemes: confidentiality attacks and authenticity attacks. Forgery is an attack that aims to falsify the authenticity of the message. Forgery attacks carried out using brute force or by exploiting weaknesses in the scheme (Schroé, 2015). These attacks are considered significant because they can invalidate the security claims of the targeted AE scheme. PAES scheme security is divided into two categories based on the nonce model used: nonce-respecting and nonce-repeating, commonly referred to as nonce-misuse. A nonce is a unique value used no more than once for the same purpose, designed to prevent replay attacks (Van Tilborg & Jajodia, 2011). In the nonce-respecting model, the attacker cannot reuse the same nonce, while in the nonce-misuse model, the attacker can use the same nonce repeatedly. Ye et al. claim that PAES-8 in the nonce-misuse model provides 128 bits of authentication security.

In 2014, Sasaki and Wang performed a forgery attack on PAES-8 (the attack was re-written in Jean et al., 2016). This forgery attack is a universal forgery conducted in the nonce-misuse model. The attack aims to generate a tag value from any plaintext of at least 15 blocks or 240 bytes. The attack exploits weakness in plaintext difference propagation through an injection process. This process uses different plaintexts to form differential attack trajectories and recover the state values in PAES-8. Sasaki and Wang stated that to produce a forgery with this method, the attacker only needs a data complexity of 2^{11} (Sasaki & Wang, 2014). However, Yu Sasaki confirmed that the attack was theoretical and had not been implemented in practice.

This gap between theoretical analysis and practical implementation introduces a research problem: Can the forgery attack on PAES-8, as proposed by Sasaki and Wang, be executed practically, and if so, how does the added complexity affect the scheme's real-world security? Understanding whether PAES-8 can withstand such attacks under nonce-misuse conditions will offer valuable insights into the scheme's resilience and inform future cryptographic designs. Therefore, in this study, an implementation of the forgery attack on PAES-8 based on the concept proposed by Sasaki and Wang is carried out using the C programming language. The aim is to determine whether the theoretical forgery attack proposed by Sasaki and Wang on the PAES-8 encryption scheme can be implemented practically and, if so, how its complexity affects the scheme's real-world security.

This study makes the following key contributions to the field of cryptographic security:

1) Practical implementation of a forgery attack:
   - This research bridges the gap between theoretical cryptanalysis and practical application by implementing the forgery attack on the PAES-8 authenticated encryption scheme, as initially proposed by Sasaki and Wang.
   - The study demonstrates that executing the attack requires modifications beyond the original theoretical model, highlighting the complexities involved in real-world scenarios.
2) Modified Attack Algorithm with Increased Complexity: The findings reveal that an additional step involving differential plaintext injection is necessary, increasing the attack complexity. This result challenges the initial assumptions about the attack's simplicity, suggesting that PAES-8 offers more resilience than previously believed.
3) Enhanced Understanding of PAES-8 Security Vulnerabilities:
   - By providing a detailed analysis of the modified forgery attack, this research contributes to a deeper understanding of the security weaknesses within PAES-8, particularly under nonce-misuse conditions.
   - The insights gained from this study lay the groundwork for further cryptanalysis efforts, especially in assessing PAES-8's security in nonce-respecting scenarios.

The present work is organized as follows: Section 2 covers PAES-8's theoretical basis and nonce-misuse security assertions. Section 3 describes how the forgery attack on PAES-8 was implemented, modifying the theory. Section 4 compares theoretical predictions with practical data and examines the attack's effects on PAES-8's security. Section 5 shows that implementing the forgery attack on PAES-8 required solving additional differential equations, increasing attack complexity and suggesting that PAES-8 may be more resilient than expected. Section 6 concludes with significant findings and research directions.

2. Literature review

2.1 Forgery Attack

Authenticity is a service that ensures the received ciphertext originates from a legitimate sender. A forgery attack targets the authenticity service within an AE scheme. A forgery attack on an AE scheme occurs when an attacker can generate a valid ciphertext/associated data/tag pair (C, AD, T), even though the scheme has never produced that pair. A brute force attack targeting authenticity is called tag guessing, where the attacker selects a ciphertext and searches for a valid tag by attempting all possible tag values. Besides brute force attacks, an attacker can also exploit weaknesses in the scheme to compromise authenticity.

There are three types of forgery attacks: existential forgery, selective forgery, and universal forgery (Liu & Liu, 2017).

1) Existential Forgery: Existential forgery occurs when an attacker can create at least one plaintext/tag pair (P, T), where P was not generated by a legitimate user. The attacker does not need control over the plaintext P, meaning the plaintext could be meaningless information.
2) Selective Forgery: Selective forgery occurs when an attacker can generate a plaintext/tag pair (P, T), where the plaintext P is chosen before the attack is executed.
3) Universal Forgery: Universal forgery occurs when an attacker can generate a plaintext/tag pair (P, T) for any given plaintext P.

Forgery attacks are based on the nonce usage model in AE schemes, specifically nonce-respecting and nonce-misuse models (Ye et al., 2014).

1) Nonce-Respecting
   In the nonce-respecting model, the nonce used in AE is guaranteed to be unique for each encryption. This restricts the attacker to using a nonce only once during a forgery attack.
2) Nonce-Misuse
   In the nonce-misuse model, the same nonce can be used more than once, which may happen due to implementation errors. When the same nonce is reused, an attacker can perform forgery attacks under the assumption that the nonce is repeated.

2.2 PAES-8

PAES requires inputs that are 128-bit key K, 128-bit nonce N, associated data AD and plaintext P of arbitrary length. The output of PAES consists of the ciphertext C and the tag value T of 128 bits. PAES consists of two structures, namely PAES-4 and PAES-8. The difference between the two structures lies in the size of the internal state used: four blocks in PAES-4 and eight blocks in PAES-8. S size, which is four blocks in PAES-4 and eight blocks in PAES-8, each measuring 128 bits.

PAES-8 consists of an internal state S which has eight words, namely $S_1, S_2, \ldots, S_8$. PAES-8 uses two similar state update functions denoted by StateUpdate_0 and StateUpdate_1. These state update functions require a block input of M of 128 bits to update the state. The two-state update functions in PAES-8 are shown in Figure 1, with StateUpdate_0 shown without the dotted line or the XOR process between S_7 and S_8. The following subsections describe the encryption and decryption of PAES-8 under forgery attacks.

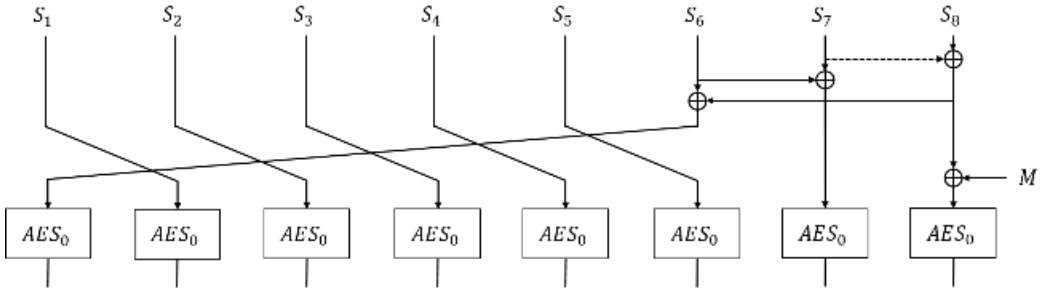

Figure 1. State Update Function in PAES-8

2.3 PAES-8 Encryption

The PAES-8 encryption scheme is divided into four stages: initialization, associated data processing, plaintext processing, and finalization, as shown in Figure 2.

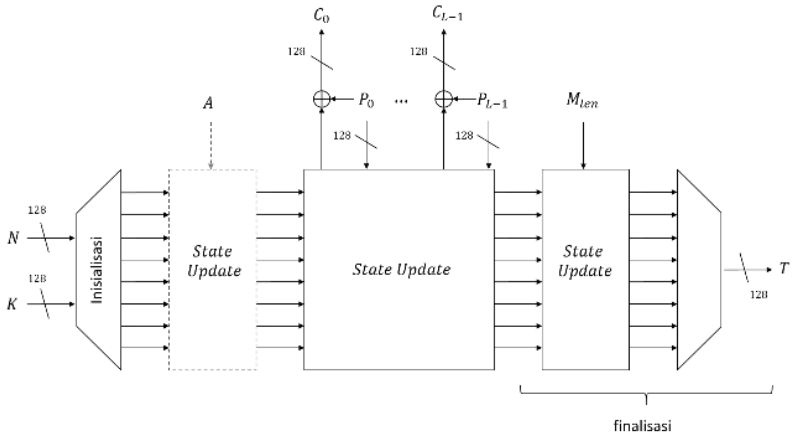

Figure 2. Overall PAES-8 Encryption Process

1) Initialization

At the initialization stage, 128 bits of the key K and 128 bits of the nonce N are mixed and loaded into eight words in the internal state. S. The state is then processed with the StateUpdate_0 function for ten rounds and XORed with the key. The initialization stage in PAES-8 is shown in Table 1. L is a linear transformation that operates on a 128-bit word $a \parallel b \parallel c \parallel d$ and is defined as $L(a, b, c, d) = (b, c, d \oplus a, a)$. L^i is the notation of the rank composition function i of the linear transformation L, e.g. $L^2 = L \circ L$.

Table 1. Algorithm 1: Initialization

Input	: 128-bit key K and 128-bit nonce N
Output	: State $S = S_1, S_2, \ldots, S_8$
1.	$S_1 = K \oplus N$
2.	$S_2 = L(K) \oplus L^3(N)$
3.	$S_3 = L^2(K) \oplus L(N)$
4.	$S_4 = L^3(K) \oplus L^2(N)$
5.	$S_5 = L^4(K) \oplus L^7(N)$
6.	$S_6 = L^5(K) \oplus L^3(N)$
7.	$S_7 = L^6(K) \oplus L^5(N)$
8.	$S_8 = L^7(K) \oplus L^6(N)$
9.	$for\ i = 1\ to\ 10$
10.	$S \leftarrow \text{StateUpdate}_0(S, 0)$
11.	$for\ i = 1\ to\ 8$
12.	$S_i \leftarrow S_i \oplus K$
13.	Return(S)

2) Plaintext Processing

A 128-bit plaintext block is defined as P_i for $i = 0, \ldots, L-1$, where L is the number of blocks in the plaintext. In each round of PAES-8, the ciphertext C_i is obtained by XORing the keystream R_i with P_i, using a single function call to StateUpdate_1. The process of plaintext encryption is illustrated in Table 2 and Figure 3.

3) Finalization

The last stage of PAES-8 encryption is finalization, which generates the *tag* value for the ciphertext. The tag value T is produced by XORing S_7 and S_8 after ten rounds of processing with StateUpdate_0 function. The input at this stage is the the plaintext length M_{len}, which is 128 bits in size. The finalization stage is shown in Table 3.

Table 2. Algorithm 2 Plaintext processing

Input	: State $S = S_1, S_2, \dots, S_8$ and plaintext P_i
Output	: Ciphertext C_i
1.	$tmp = S_7$
2.	$S \leftarrow \text{StateUpdate}_1(S, P_i)$
3.	$R_i = tmp \oplus S_7$
4.	$C_i = P_i \oplus R_i$
5.	Return(S)

Table 3. Algorithm 3 Finalization

Input	: State $S = S_1, S_2, \dots, S_8$, plaintext length M_{len}
Output	: Tags T
1.	for $i = 1$ to 10
2.	$S \leftarrow \text{StateUpdate}_0(S, M_{len})$
3.	$T = S_7 \oplus S_8$
4.	Return(T)

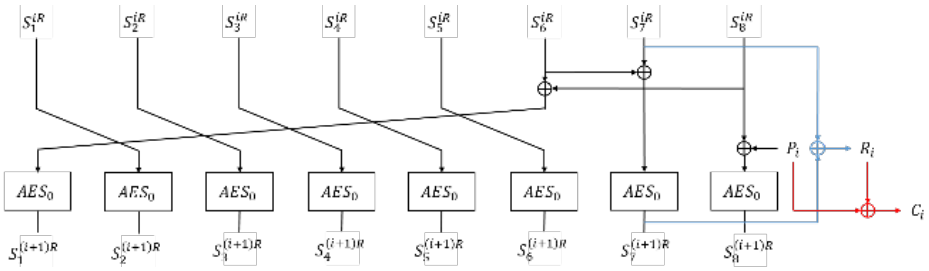

Figure 3. Processing of One Block ofPlaintext on PAES-8

2.4 *Decryption on PAES-8*

The initialization and processing stages of associated data in ciphertext decryption are the same as in plaintext encryption. Each ciphertext block C_i is process, as shown in Figure 4. The entire internal state is updated, except S_8 using the function StateUpdate_1. The plaintext P_i is obtained through the XOR operation between the keystream R_i with the ciphertext C_i and then S_8 is updated.

Tag value generation during decryption is the same as in encryption. In decryption, if the generated tag value is equal to T, the plaintext P is obtained; otherwise, the error symbol $\perp$ is returned.

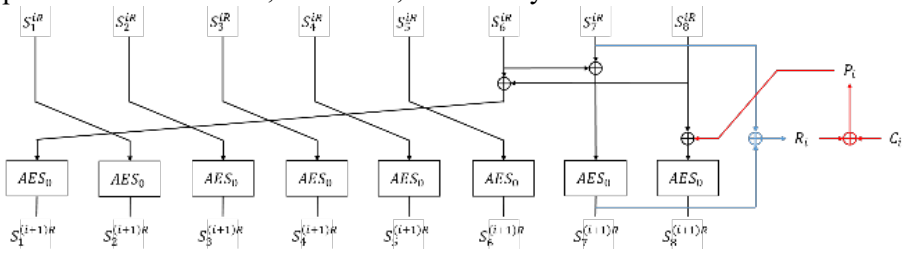

Figure 4. Processing of One Ciphertext Block on PAES-8

2.5 *Sasaki and Wang's Forgery Attack on PAES-8*

Sasaki and Wang's forgery attack on PAES-8 constitutes a universal forgery. The objective of this attack is to generate valid tag values for any plaintext consisting of at least 15 blocks (240 bytes). The attack leverages state recovery, meaning that once the internal states of a specific round are fully known, the remaining finalization steps can be executed efficiently, allowing the tag values to be derived independently.

The forgery attack on PAES-8 is carried out using the nonce-misuse model, where the same nonce in each message encryption. Additionally, to simplify the attack, Sasaki and Wang (2014) set the associated data to an

empty string; in other words, no associated data is used during the plaintext encryption process. Based on these two assumptions, the encryption process for each plaintext produces the same state value during the initialization stage. After this stage, the process continues with the plaintext processing stage.

Sasaki and Wang's forgery attack on PAES-8 utilizes differential trajectories, illustrated by the thick red lines in Figure 5. These trajectories are obtained from the encryption process of the forgery target P and the injection of different plaintexts, ΔP_α on the plaintext block P_0 and ΔP_β on P_1, with cancellation occuring between them in S_8. The differential attack trajectory highlights the state blocks with non-zero difference values between the forgery target encryption P and plaintext injection P'.

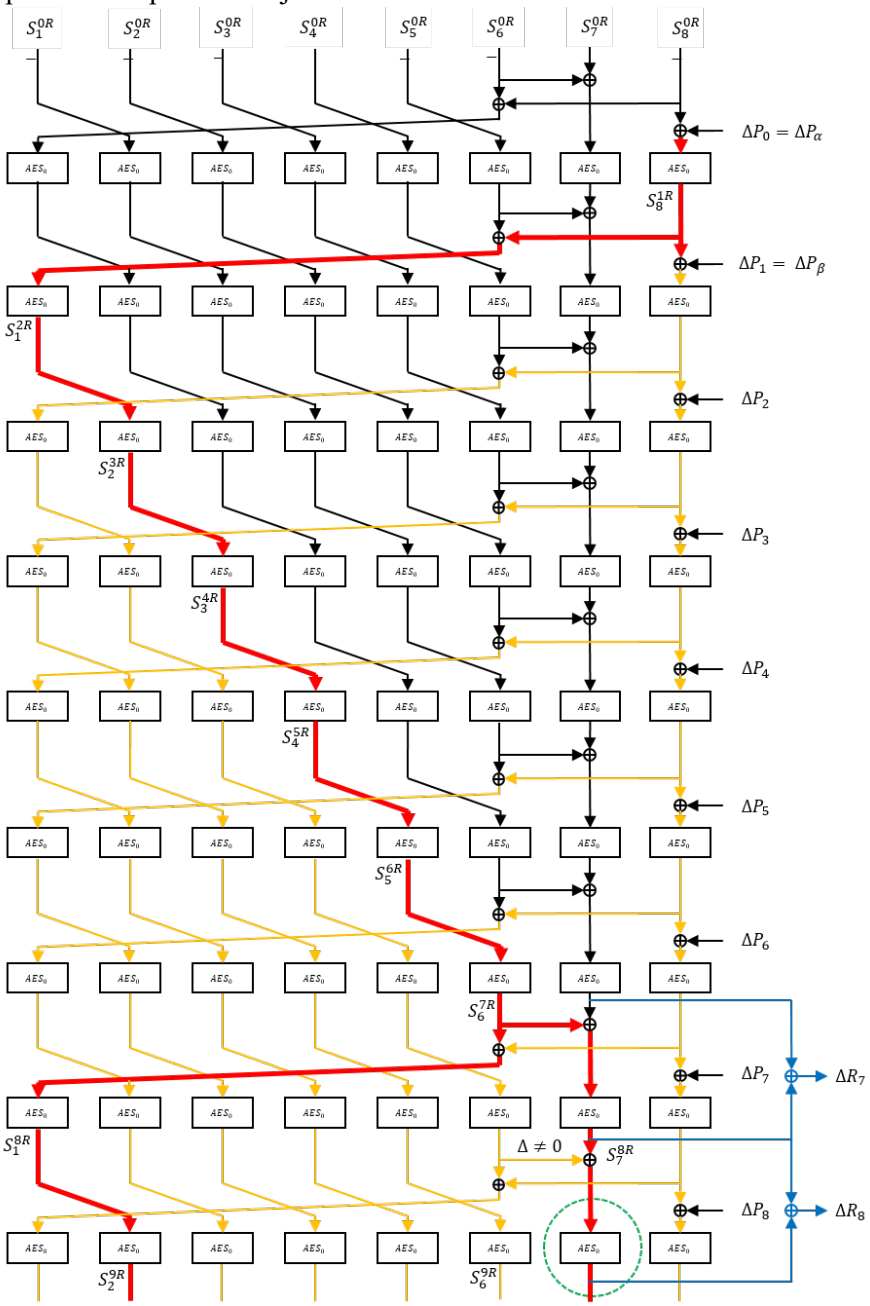

Figure 5. Differential Trajectory of the Injection Process Results

The formation of the differential trajectory can result in two cases: the case of cancellation and the case of no cancellation. Figure 5 illustrates the two differential trajectories formed during the injection process. The thick

red line represents the differential trajectory when cancellation occurs, while the absence of cancellation is indicated by an additional orange trajectory. Based on these two cases, the cancellation of each injection process is detected to determine whether a differential trajectory of the attack is formed. This attack differential trajectory is then used during the state recovery stage.

The steps in this attack can be summarized as follows:

a. Injection of different plaintexts into two consecutive plaintext blocks such that cancellation occurs in S_8 with high probability.
b. The invalidation of S_8 is indicated by a different ciphertext after the eighth round; if this occurs, it will leak information about the state.
c. Once the state is restored, tags are produced by going through the remaining transformations of what is now the public construction.

The forgery attack on PAES-8 for plaintext $P = (P_0 \parallel P_1 \parallel \cdots \parallel P_{14})$ is shown in Algorithm 4. In Algorithm 4, there are two iterations. The first iteration aims to recover S_8 and S_7 over five rounds, while the second iteration aims to recover S_7 over the next two rounds. Each iteration will produce two pairs, ΔP_α and ΔP_β, with a probability of 2^{-6}. In this case, if no canceling differential trajectory is found with the probability of 2^{-6}, the attacker can replace it with a probability of 2^{-7} and run the loop 2^8 times. Based on Table 4, the attack complexity is: $16.2^7 + 2^7 \approx 2^{11}$ computations.

Table 4. Algorithm 4 Sasaki and Wangs' Universal Forgery Attack

Input	: Plaintext $P = (P_0 \parallel P_1 \parallel \cdots \parallel P_{14})$
Output	: Ciphertext $C = C_0 \parallel C_1 \parallel \cdots \parallel C_{14})$ and tags T
1.	Query the first 15 plaintext blocks of the targe $P = (P_0 \parallel P_1 \parallel \cdots \parallel P_{14})$t and get the key stream $R_0, R_1, \dots, R_{14}$.
2.	$\boldsymbol{for}\ pos = 1\ \boldsymbol{to}\ 16\ \boldsymbol{do}$
3.	$\boldsymbol{for}\ k = 1\ \boldsymbol{to}\ 2^7\boldsymbol{do}$
4.	Select a different plaintext ΔP_α^k with 1 byte active at the specified position and find ΔP_β^k that corresponds.
5.	Query $(P_0 \oplus \Delta P_\alpha^k \parallel P_1 \oplus \Delta P_\beta^k \parallel P_2 \parallel \cdots \parallel P_{14})$ and obtain the keystream $R_0^k, \dots, R_{14}^k$.
6.	Check if the difference $R_7 \oplus R_7^k$ can produce $R_7 \oplus R_7^k \oplus R_8 \oplus R_8^k$ with AES_0.
7.	Verify the same property in the next 4 rounds.
8.	Keep the pairs that satisfy steps 6 and 7.
9.	$\boldsymbol{end\ for}$
10.	Recover the byte at pos from the word state S_8 in loop 0.
11.	$\boldsymbol{end\ for}$
12.	Recover S_7 from rounds 8, 9, 10, 11, and 12.
13.	$\boldsymbol{for}\ k = 1\ \boldsymbol{to}\ 2^7\ \boldsymbol{do}$
14.	Select the difference ΔP_α^k with 1 byte active at a specified position and find ΔP_β^k that corresponds.
15.	Query $(P_0 \parallel P_1 \parallel P_2 \oplus \Delta P_\alpha^k \parallel P_3 \oplus \Delta P_\beta^k \parallel \cdots \parallel P_{14})$ and obtain the key stream $R_0^k, \dots, R_{14}^k$.
16.	Check if the difference $R_9 \oplus R_9^k$ can produce $R_9 \oplus R_9^k \oplus R_{10} \oplus R_{10}^k$ with AES_0.
17.	Verify the same property in the next 4 rounds.
18.	Save the pairs that satisfy steps 16 and 17.
19.	end for
20.	Recover S_7 from laps 13 and 14.
21.	Determine all states in round 8.
22.	Continue the rest of the transformation and generate the tag value.

3. Method

3.1 Experiment Setup

The experiment was conducted on a laptop featuring an Intel Core i3 processor, dual-core, running at 2.00 GHz and 8 GB of RAM. The device's operating system supported the C programming language, and the code

was developed using the GCC or Clang compilers for portability. Optional integrated development environments (IDEs), such as Visual Studio Code or Code::Blocks were utilized to streamline coding and debugging. The implementation required cryptographic functions, which were achieved either through custom AES code or by leveraging libraries like OpenSSL. This setup ensures that the computational resources and software tools were adequate to efficiently perform encryption, injection, and state recovery processes.

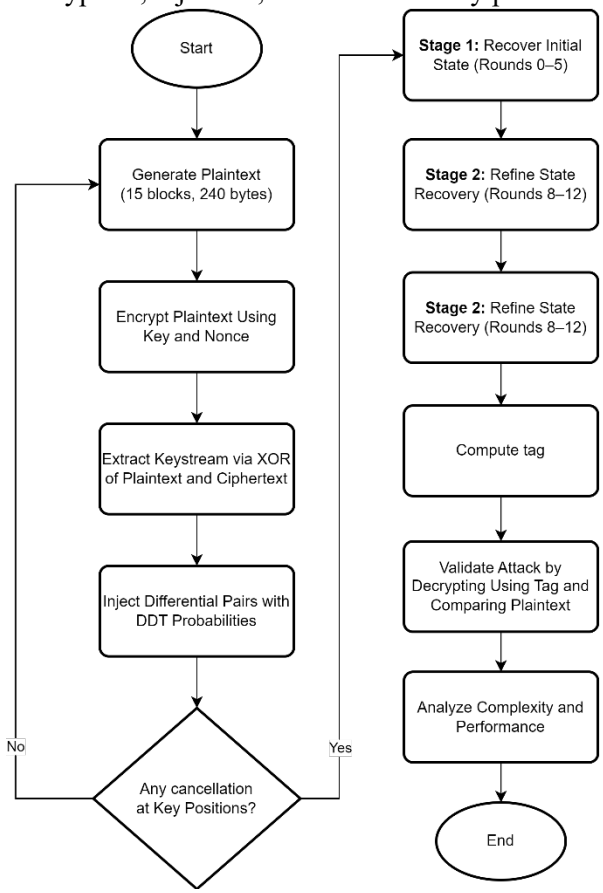

Figure 6. Research Stages

3.2 Research Stages

In this attack, it is assumed that the attacker can perform encryption requests. These encryption requests are made for each different plaintext using the same key and nonce, without incorporating associated data. The forgery attack is implemented through the following stages (see Figure 6): .

a. Generating the forgery target, which is the plaintext $P = (P_0 \parallel P_1 \parallel \cdots \parallel P_{14})$ of 15 blocks or 240 byte*s,* as well as obtaining the ciphertext $C = (C_0 \parallel C_1 \parallel \cdots \parallel C_{14})$ through the encryption process and obtaining the keystream $R_0, R_1, \ldots, R_{14}$ by performing an XOR operation between the plaintext and the ciphertext.
b. Generating the Differential Distribution Table (DDT) from the AES s-box.
c. Performing different selections of plaintext ΔP_α and ΔP_β using the difference (α, β) with a probability of 2^{-6}, based on the following equation:

$\Delta P_\alpha = (\alpha, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$ .....1)
$\Delta P_\beta = MixColumns \circ ShiftRows\ (\beta, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$.....2)

with α placed in any 16-byte position in the state and β in the same position as α.

d. Performing the injection of ΔP_α and ΔP_β into the plaintext block P_0 and P_1.
e. Detecting any cancellation that occurs between ΔP_α and ΔP_β at S_8.
f. Recovering the internal state based on the detected differential trajectories.
g. Computing the tag value for the ciphertext based on the recovered state.

h. Validating attack by decrypting using the tag and comparing the plaintext.
i. Analyzing complexity and performance.

3.3 Population and Sample

The forgery target used in this attack implementation consists of 15 blocks (240 bytes) of plaintext, denoted as $P = (P_0 \parallel P_1 \parallel \cdots \parallel P_{14})$. The choice of 15 blocks is based on the minimum requirement for the forgery attack, as specified by Sasaki and Wang. Consequently, the attack population comprises $2^{240\times8} = 2^{1920}$ possible plaintexts. In this study, only one plaintext is selected as the forgery target, as the attack is classified as a universal forgery, meaning it can be executed on any given plaintext. The forgery target is randomly generated and is presented in Table 5.

Table 5. Plaintext P Used in the Forgery Attack Implementation

P_i	Plaintext Block
P_0	$0xe6ff57aabd694a18a9424a25e70979c8$
P_1	$0xde72a92c8a5454ac70be9a4aabc26232$
P_2	$0x8b28f2643260a6cac08248de76a82cee$
P_3	$0xbc0cc6350202cd91d4bbd2c6c1f2aa7e$
P_4	$0xb3bef2426b3ab4c8c0f42613118f5076$
P_5	$0x9c3c65cd0eedd5aff4494652fdf5efc9$
P_6	$0x24f453893fb126c22140c19252fd908e$
P_7	$0xb5662aec53fe8be22aa92d89f66b9fb2$
P_8	$0xb3af77f352c0ff6995eefc70540351df$
P_9	$0x19a4077f0f2ba45033504488ca458eb8$
P_{10}	$0x76471e6759c4e59b67878243614d0c7d$
P_{11}	$0xa90af7bbee9e066492fbfdb8a3fe64eb$
P_{12}	$0x5f2ec5890892a9faefc69e58fbfd945c$
P_{13}	$0xbb2a3eec9edb979aada571ae07774909$
P_{14}	$0xdcf256cadc817502a037018058968f73$

Table 6. Ciphertext C Used in the the Forgery Attack Implementation

C_i	Ciphertext Block
C_0	$0x18edd99b8612aef2a02e422f298f4895$
C_1	$0x767efbe40c51638b94ce26b289cec7b9$
C_2	$0x6a199e0d40e8a041f2b73781d394d811$
C_3	$0x56fd9c5fa7fe9abceba780d75313cb5b$
C_4	$0xa086a539824fc9cb7622256de53fb109$
C_5	$0x10fa4ff26b4b09d674eb4708cce4b888$
C_6	$0x3a8903cf2e4499cdac631e83656122dd$
C_7	$0x58b342a2290baa45d5cf7c819f692c77$
C_8	$0x4072bcbe0b5de414ad7813d25a8ce8de$
C_9	$0x5c01d1488b9b9e2d3e92bd3bb3c8d0d4$
C_{10}	$0xb40689668c641a82e0197c7993548a7d$
C_{11}	$0xc12dce701108fc4561a4a67f64116b98$
C_{12}	$0xf346e62bcb67fc9c58d4df62f5cdc4f0$
C_{13}	$0x195a68f43e0aad84eac630a3e20c6e05$
C_{14}	$0x71327d9f47c6906c2cf70d2b3f8e7d6e$

The message length of the forgery target is defined by $M_{len} = 0x00000000000000000000000000000780$, which represents 1920 bits. The forgery target is subjected to an encryption request using a key and nonce with values of $K = 0x10cd91f4cf0d3d54a3301fa939fac4fc$ and $N = 0x36283b17d3ec71002373ff5e6d8309a6$. The key and nonce are randomly generated from a population equal to 2^{128}. The encryption result of the plaintext P, which is the ciphertext C , is shown in Table 6.

4. Results

This section describes the implementation results of Sasaki and Wang's (2014) forgery attack on PAES-8. The explanation of the attack implementation consists of the injection of different plaintexts, ΔP_α and ΔP_β, state recovery, and tag value generation. In state recovery, it is known that recovery S_8 requires a differential attack trajectory formed from the cancellation of the $(\Delta P_\alpha, \Delta P_\beta)$ with a probability of 2^{-6} and $(\Delta P_\alpha, \Delta P_\beta)$ with a probability of 2^{-7}. Therefore, during the injection process, the selection and injection of different plaintexts, ΔP_α and ΔP_β, are performed withprobability 2^{-6} and 2^{-7}, respectively.

4.1 Plaintext Differential Injection ΔP_α and ΔP_β

The injection stage of the Sasaki and Wang's forgery attack aims to obtain the differential trajectory of the attack. This is achieved by performing an XOR operation with different plaintexts. Specifically, ΔP_α is applied to the plaintext block P_0, and the light text difference ΔP_β is applied to the light text block P_1. The injection of ΔP_α is performed to cancel the effect of the ΔP_β injection on the activation S_8, thereby forming a differential attack trajectory. The following subsections describe the results of selecting ΔP_α and ΔP_β during the injection stage, as well as the detection of cancellations occuring between the injections.

4.1.1 Differential Selection of Plaintext ΔP_α and ΔP_β

The Selection of ΔP_α and ΔP_β is based on the input and output difference values derived from DDT of the AES s-box. All differences (α, β) with DDT values of 4 or 2 are selected. Table 7 and Table 8 show the (α, β) differences with probabilities of 2^{-6} and 2^{-7}, respectively.

Table 7. Differences (α, β) with a DDT value of 4

No.	(α, β)
1.	$(0x01, 0x1f)$
2.	$(0x02, 0x14)$
⋮	⋮
254.	$(0xfe, 0xd8)$
255.	$(0xff, 0x75)$

Table 8. Differences (α, β) with a DDT value of 2

No.	(α, β)
1.	$(0x01, 0x01)$
2.	$(0x01, 0x04)$
⋮	⋮
32.129.	$(0xff, 0xfe)$
32.130.	$(0xff, 0xff)$

Based on Table 7 and Table 8, there are 255 differences (α, β) with a DDT value of 4 and a total of 32,130 differences (α, β) with a DDT value of 2. From these, $2^7 = 128$ differences (α, β) with DDT a value of 4 and $2^8 = 256$ differences $(\alpha. \beta)$ with a DDT value of 2 are used as ΔP_α and ΔP_β pairs in the injection process.

4.1.2 Cancellation Detection between ΔP_α and ΔP_β

The detection result for the invalidation of 2,048 injections of ΔP_α and ΔP_β with a cancellation probability of 2^{-6} and $16 \times 2^8 = 4.096$ injection of ΔP_α and ΔP_β with a cancellation probability of 2^{-7}, are shown in Table 9 and Table 10. These tables display the (α, β) differences that form ΔP_α and ΔP_β at the 16 positionscausing the invalidation.

Table 9. Differences (α, β) with Probability 2^{-6} Causing Cancellation

No.	Differences (α, β) at the i-th Position			
	1	2	3	4
1.	$(0x5b, 0x5a)$	$(0x0f, 0x15)$	$(0x0c, 0x9d)$	$(0x34,0x7b)$
2.	–	–	$(0x51, 0xb2)$	–
3.	–	–	$(0x5d, 0x2f)$	–
	5	6	7	8
1.	$(0x39, 0x71)$	$(0x5e, 0x3b)$	$(0x3f, 0x16)$	$(0x55, 0x9f)$
2.	$(0x4d, 0x80)$	–	–	–
3.	$(0x74, 0xf1)$	–	–	–
	9	10	11	12
1.	$(0x6b, 0x1c)$	$(0x6b, 0x1c)$	$(0x17, 0x93)$	$(0x42, 0x4f)$
2.	–		$(0x2c, 0x12)$	
3.	–		$(0x3b, 0x81)$	
	13	14	15	16
1.	$(0x52, 0x63)$	$(0x10, 0xa9)$	$(0x77, 0x96)$	$(0x59, 0xa8)$
2.	–	$(0x25, 0x5c)$	–	–
3.	–	$(0x35, 0xf5)$	–	–

Based on Table 9 and Table 10, it is evident that 24 out of a total of the 2,048 injections with a cancellation probability of 2^{-6} and 32 out of a total of 4,096 injections with a probability of 2^{-7}, results incancellation. Theseinvalidations occur accross all 16 positions of ΔP_α and ΔP_β.

The invalidation results in Table 9 and Table 10 also show invalidation for all positions. For injections with a cancellation probability of 2^{-6}, among the 16 positions, at most three injections and at least one injection cause cancellations. In contrast, for injections with a cancellation probability of 2^{-7}, the number of cancelations is consistent across all positions, woth two injections per position. The ΔP_α and ΔP_β formed from the differing (α, β) values causing the cancellations are collectively referred to as the *Cancellation* $(\Delta P_\alpha, \Delta P_\beta$

Table 10. Differences (α, β) with Probability 2^{-7} Causing Cancellation

No.	Differences (α, β) at the i-th Position			
	1	2	3	4
1.	$(0x01, 0x11)$	$(0x01, 0xdd)$	$(0x01, 0x06)$	$(0x01, 0xa8)$
2.	$(0x02, 0xb1)$	$(0x02, 0xa1)$	$(0x02, 0x83)$	$(0x02, 0xfa)$
	5	6	7	8
1.	$(0x01, 0x0f)$	$(0x01, 0x97)$	$(0x01, 0xf7)$	$(0x01, 0xac)$
2.	$(0x02, 0xaa)$	$(0x02, 0x12)$	$(0x02, 0xe6)$	$(0x02, 0xfa)$
	9	10	11	12
1.	$(0x01, 0x80)$	$(0x01, 0x2a)$	$(0x01, 0xa9)$	$(0x01, 0x59)$
2.	$(0x02, 0x85)$	$(0x02, 0xfd)$	$(0x02, 0x40)$	$(0x02, 0x8e)$
	13	14	15	16
1.	$(0x01, 0xff)$	$(0x01, 0x8e)$	$(0x01, 0x93)$	$(0x01, 0xa1)$
2.	$(0x02, 0xde)$	$(0x02, 0x0c)$	$(0x02, 0x2f)$	$(0x02, 0xf2)$

4.2 State Recovery

The state is recovered by observing the obtained differential trajectories and the trajectories in the PAES-8 encryption scheme. There are no specific stipulations on the selection of attack differential trajectories, but any attack differential trajectory that align the thick red line in Figure 5 can be used. State recovery is conducted in stages.

1) Recovery of S_8^{8R}

The recovery of S_8^{8R} is achieved by first recovering S_8 in the 0th round (S_8^{0R}) and then progressively recovering S_8 for all rounds up to S_8^{8R}. The recovery of S_8^{0R} is performed byte by byte using 16 differential attack trajectories obtained from the injection of Cancellation $(\Delta P_\alpha, \Delta P_\beta)$ at each position. Table 11 and Table 12 show the solutions to the differential equations for recovery at 16 positions of S_8^{0R}. The solutions in Table 11 uses the Cancellation $(\Delta P_\alpha, \Delta P_\beta)$ probability 2^{-6}, derived from the difference (α, β) in Table 9. Meanwhile, the solutions in Table 12 uses the canceling probability formed from the difference in Table 9 $(\Delta P_\alpha, \Delta P_\beta)$ probability 2^{-7}, derived from the difference (α, β) shown in Table 10.

Table 11. Solution of the differential equation for the recovery of S_8^{0R} using Cancellation $(\Delta P_\alpha, \Delta P_\beta)$ probability 2^{-6}

Position	(α, β)	Solution
1	$(0x5b, 0x5a)$	$0x00, 0x5b, 0xa9, 0xf2$
2	$(0x0f, 0x15)$	$0x00, 0x0f, 0x81, 0x8e$
3	$(0x0c, 0x9d)$	$0x00, 0x0c, 0x51, 0x5d$
4	$(0x34,0x7b)$	$0x00, 0x34, 0x99, 0xad$
5	$(0x39, 0x71)$	$0x00, 0x39, 0x4d, 0x74$
6	$(0x5e, 0x3b)$	$0x00, 0x5e, 0x8d, 0xd3$
7	$(0x3f, 0x16)$	$0x00, 0x3f, 0xd1, 0xee$
8	$(0x6b, 0x1c)$	$0x00, 0x55, 0x9b, 0xce$
9	$(0x6b, 0x1c)$	$0x00, 0x6b, 0x9d, 0xf6$
10	$(0x6b, 0x1c)$	$0x00, 0x6b, 0x9d, 0xf6$
11	$(0x17, 0x93)$	$0x00, 0x17, 0x2c, 0x3b$
12	$(0x42, 0x4f)$	$0x00, 0x42, 0xb7, 0xf5$
13	$(0x52, 0x63)$	$0x00, 0x52, 0x82, 0xd0$
14	$(0x10, 0xa9)$	$0x00, 0x10, 0x25, 0x35$
15	$(0x77, 0x96)$	$0x00, 0x77, 0x92, 0xe5$
16	$(0x59, 0xa8)$	$0x00, 0x59, 0x93, 0xca$

Table 12. Solution of the differential equation on recovery of S_8^{0R} using the Cancellation $(\Delta P_\alpha, \Delta P_\beta)$ probability 2^{-7}

Position	(α, β)	Solution
1	$(0x01, 0x11)$	$0xa8, 0xa9$
2	$(0x01, 0xdd)$	$0x0e, 0x0f$
3	$(0x01, 0x06)$	$0x5c, 0x5d$
4	$(0x01,0xa8)$	$0x98, 0x99$
5	$(0x01, 0x0f)$	$0x74, 0x75$
6	$(0x01, 0x97)$	$0x5e, 0x5f$
7	$(0x01, 0xf7)$	$0xee, 0xef$
8	$(0x01, 0xac)$	$0x9a, 0x9b$
9	$(0x01, 0x80)$	$0x9c, 0x9d$
10	$(0x01, 0x2a)$	$0xf6, 0xf7$
11	$(0x01, 0xa9)$	$0x2c, 0x2d$
12	$(0x01, 0x59)$	$0xf4, 0xf5$
13	$(0x01, 0xff)$	$0x82, 0x83$
14	$(0x01, 0x8e)$	$0x34, 0x35$
15	$(0x01, 0x93)$	$0x92, 0x93$
16	$(0x01, 0xa1)$	$0x58, 0x59$

Based on Tables 11 and Table 12, it is known that there is one solution from both tables that has the same value. The red-colored solution in Table 12 indicates the same value in both differential equation solutions. Thus, it can be concluded that the overall 128-bit differential equation solution is 0xa90f5d99745eee9b9df62cf582359259.

The solution value of the differential equation is the input value AES_0 at round 0; in other words, it is the value of the $S_8^{0R} \oplus P_0$. Therefore, to recover S_8^{0R}, the calculation $S_8^{0R} \oplus P_0 \oplus P_0 = 0x4ff00a33c937a48334b466d0653ceb91$ is performed. Subsequently, the recovery of S_8^{8R} is carried out, yielding $S_8^{8R} =$ 0xa7617d5028c0d68dd81588dcecd050d5.

2) Recovery $S_7^{8R}, S_7^{9R}, S_7^{10R}, S_7^{11R}$, and S_7^{12R}

Recovery of S_7 uses a plaintext difference formed from the difference $(\alpha, \beta) = (0x5b, 0x5a)$ at position 1 and $(\alpha, \beta) = (0x0f, 0x15)$ at position 2 with the plaintext difference values shown in Table 13.

Table 13. Plaintext Differences for the Recovery of S_7

Plaintext Differences	Value
$(\Delta P_{\alpha 1}^1, \Delta P_{\beta 1}^1)$	$0x5b000000000000000000000000000000$
	$0xb45a5aee00000000000000000000000$
$(\Delta P_{\alpha 1}^2, \Delta P_{\beta 1}^2)$	$0x000f0000000000000000000000000000$
	$0x0000000000000000000000003f2a1515$

The recovery of S_7 is performed by solving 16 differential equations of the form $S(x) \oplus S(x \oplus \Delta input) = \Delta output$. Table 14 shows the solutions to the differential equations for the recovery S_7^{8R} in each byte.

Table 14. Differential Equation Solutions for the Recovery of S_7^{8R}

The i-th byte	Solution		Conclusion
	$(\Delta P_{\alpha 1}^1, \Delta P_{\beta 1}^1)$	$(\Delta P_{\alpha 1}^2, \Delta P_{\beta 1}^2)$	
1	$0x13, 0xa6$	$0x86, 0xa6$	$0xa6$
2	$0x4a, 0xf3$	$0x0a, 0xf3$	$0xf3$
3	$0x96, 0xb0$	$0x0d, 0x96$	$0x96$
4	$0x82, 0xc5$	$0x15, 0xc5$	$0xc5$
5	$0x09, 0xd1$	$0x09, 0x34$	$0x09$
6	$0x36, 0xcb$	$0x52, 0xcb$	$0xcb$
7	$0x25, 0x61$	$0x25, 0xa0$	$0x25$
8	$0xa5, 0xa8$	$0x7f, 0xa5$	$0xa5$
9	$0x1d, 0x31$	$0x31, 0xc7$	$0x31$
10	$0x36, 0x85$	$0x5c, 0x85$	$0x85$
11	$0x12, 0x8c$	$0x12, 0x40$	$0x12$
12	$0x47, 0xd7$	$0xa0, 0xd7$	$0xd7$
13	$0x62, 0x7e$	$0x4d, 0x62$	$0x62$
14	$0x54, 0xcf$	$0xcf, 0xd9$	$0xcf$
15	$0x52, 0xd5$	$0xd5, 0xdb$	$0xd5$
16	$0x74, 0xfc$	$0x05, 0xfc$	$0xfc$

Based on Table 14, the correct solution to the 16 differential equations is obtained, namely 0xa6f396c509cb25a5318512d762cfd5fc which is the value of $S_6^{8R} \oplus S_7^{8R}$. Thus, we get $S_7^{8R} =$ 0xe337b28c5c0a7abdbef393e6677072ca.In the same way, S_7 is restored for the next 4 rounds.

Table 15 shows the 128-bit differential equation solutions for the recovery of $S_7^{9R}, S_7^{10R}, S_7^{11R}$ and S_7^{12R}. Table 16 shows the recovery values for $S_7^{9R}, S_7^{10R}, S_7^{11R}$ and S_7^{12R}. The values of S_7 are used to recover other state*s* in the 8th round. Table 17 shows the recovery results for S_6^{8R} down to S_3^{8R} based on the values of S_7.

Table 15. Solutions to 16 Differential Equations for the Recovery of $S_7^{9R}, S_7^{10R}, S_7^{11R}$ and S_7^{12R}

State	Solution to 16 Differential Equations
$S_6^{9R} \oplus S_7^{9R}$	$0x1276dd532a0bd9278f26d2657c54f49b$
$S_6^{10R} \oplus S_7^{10R}$	$0xaed46da82f03169db9a7b00c27205ec3$
$S_6^{11R} \oplus S_7^{11R}$	$0x06e1af63afef6b819f18a7bde7deb638$
$S_6^{12R} \oplus S_7^{12R}$	$0x09c792e8ad6b3d4d78fdb216f313d1b0$

Table 16. Recovery Results for $S_7^{9R}, S_7^{10R}, S_7^{11R}$ and S_7^{12R}

State	State Recovery Value
S_7^{9R}	$0x10ea79c1059761c086657c4469ffcbcb$
S_7^{10R}	$0x554faff681275bbd8ba785f7107295a7$
S_7^{11R}	$0x970e38f75487a4a40c397bcde26b13a7$
S_7^{12R}	$0xff29013cab115e85ff66200a25841cd4$

Table 17. Recovery Results for $S_6^{8R}, S_5^{8R}, S_4^{8R}$ and S_3^{8R}

State	State Recovery Value
S_6^{8R}	$0x45c4244955c15f188f76813105bfa736$
S_5^{8R}	$0x9b4679b977cc5ced3a996934e5c5f1d7$
S_4^{8R}	$0x4a9452dd90fdb9e9de253912f1d86ed4$
S_3^{8R}	$0x7965b69a35155c91847c566911e416d3$

3) Recovery S_7^{13R} and S_7^{14R}

The recovery of S_7^{13R} is conducted by repeating the attack, namely injecting different plaintexts ΔP_α and ΔP_β into the plaintext block P_2 and P_3. Table 18 and Table 19 display the values of the differences (α, β) forming ΔP_α and ΔP_β at the 16 positions, which led to the cancellation of the attack repetition.

Table 18. Difference (α, β) Probability 2^{-6} That Causes Abortions in Attack Repetition

No.	Difference (α, β) at the i-th Position			
	1	2	3	4
1.	$(0x30, 0x67)$	$(0x28, 0x57)$	$(0x65, 0x2e)$	$(0x7e, 0x90)$
2.	$(0x5f, 0xac)$	–	–	–
3.	$(0x6f, 0xcb)$	–	–	–
	5	6	7	8
1.	$(0x42, 0x4f)$	$(0x42, 0x4f)$	$(0x0c, 0x9d)$	$(0x59, 0xa8)$
2.	–	–	$(0x51, 0xb2)$	–
3.	–	–	$(0x5d, 0x2f)$	–
	9	10	11	12
1.	$(0x3d, 0x44)$	$(0x5e, 0x3b)$	$(0x1b, 0xcc)$	$(0x0f, 0x15)$
2.	–		$(0x66, 0x50)$	
3.	–		$(0x7d, 0x9c)$	
	13	14	15	16
1.	$(0x17, 0x93)$	$(0x0b, 0x48)$	$(0x1b, 0xcc)$	$(0x2e, 0x52)$
2.	$(0x2c, 0x12)$	$(0x43, 0x79)$	$(0x66, 0x50)$	$(0x58, 0x09)$
3.	$(0x3b, 0x81)$	$(0x48, 0x31)$	$(0x7d, 0x9c)$	$(0x76, 0x5b)$

Table 19. Difference (α, β) Probability 2^{-7} That Cause Abortions in Attack Repetition

No.	Difference (α, β) at the i-th Position			
	1	2	3	4
1.	$(0x01, 0x97)$	$(0x01, 0x4b)$	$(0x01, 0xe6)$	$(0x01, 0x3a)$
2.	$(0x02, 0x83)$	$(0x02, 0x0d)$	$(0x02, 0xc8)$	$(0x02, 0xbc)$
	5	6	7	8
1.	$(0x01, 0x59)$	$(0x01, 0x59)$	$(0x01, 0x29)$	$(0x01, 0x6b)$
2.	$(0x02, 0x8e)$	$(0x02, 0x8e)$	$(0x02, 0x55)$	$(0x02, 0x9c)$
	9	10	11	12
1.	$(0x01, 0x5a)$	$(0x01, 0xd3)$	$(0x01, 0xef)$	$(0x01, 0xdd)$
2.	$(0x02, 0xd0)$	$(0x02, 0x58)$	$(0x02, 0x2d)$	$(0x02, 0xa1)$
	13	14	15	16
1.	$(0x01, 0xa9)$	$(0x01, 0x4c)$	$(0x01, 0xb6)$	$(0x01, 0x24)$
2.	$(0x02, 0x40)$	$(0x02, 0x2a)$	$(0x02, 0x70)$	$(0x02, 0x40)$

The recovery of S_7^{13R} and S_7^{14R} is conducted by solving differential equations. Two attack differential trajectories formed from plaintext difference injection are used, as shown in Table 20. $(\Delta P_{\alpha 1}^1, \Delta P_{\beta 1}^1)$ and Cancellation $(\Delta P_{\alpha 2}^2, \Delta P_{\beta 2}^2)$ are differential plaintexts formed from the differences $(\alpha, \beta) = (0x30, 0x67)$ and $(\alpha, \beta) = (0x5f, 0xac)$ at position 1. Table 21 shows the recovery results for S_7^{13R} and S_7^{14R}, while Table 22 shows the recovery results for S_2^{8R} and S_1^{8R}, based on recovery S_7^{13R} and S_7^{14R}.

Table 20. Plaintext Differences for the Recovery of S_7^{13R} and S_7^{14R}

Plaintext Differences	Value
$(\Delta P_{\alpha 2}^1, \Delta P_{\beta 2}^1)$	$0x30000000000000000000000000000000$
	$0xce6767a900000000000000000000000$
$(\Delta P_{\alpha 2}^2, \Delta P_{\beta 2}^2)$	$0x5f000000000000000000000000000000$
	$0x43acacef00000000000000000000000$

Table 21. Recovery Results for S_7^{13R} and S_7^{14R}

State	State Recovery Value
S_7^{13R}	$0x5341229e68e40be3487461302bb44c78$
S_7^{14R}	$0xf1317486c83531fd0f17203dcecf6b74$

Table 22. Recovery Results for S_2^{8R} and S_1^{8R}

State	State Recovery Value
S_2^{8R}	$0xa69468d23637c058b18c04dbba25b685$
S_1^{8R}	$0x61d53827717ddeccf34dfac048c72b85$

4.3 Tag Value Generation

Tag value generation begins by processing the forgery target's plaintext after the 8th round, i.e. $P_8 \parallel P_9 \parallel P_{10} \parallel P_{11} \parallel P_{12} \parallel P_{13} \parallel P_{14}$ based on Algorithm 3 and using the recovered 8th round state values. The results of all 8th round state recoveries are given in Table 23.

This plaintext processing produces the ciphertext and state value resulting from the processing of the last plaintext block or the 15th round state value. The state value is then used to generate the tag value based on Algorithm 4. The 15th round state value is provided in Table 24. The tag value generation results is $0xcb28ff49c79c5db4361064163f36aeaa$.

Table 23. State recovery results in the 8th round

State	State Recovery Value
S_8^{8R}	$0xa7617d5028c0d68dd81588dcecd050d5$
S_7^{8R}	$0xe337b28c5c0a7abdbef393e6677072ca$
S_6^{8R}	$0x45c4244955c15f188f76813105bfa736$
S_5^{8R}	$0x9b4679b977cc5ced3a996934e5c5f1d7$
S_4^{8R}	$0x4a9452dd90fdb9e9de253912f1d86ed4$
S_3^{8R}	$0x7965b69a35155c91847c566911e416d3$
S_2^{8R}	$0xa69468d23637c058b18c04dbba25b685$
S_1^{8R}	$0x61d53827717ddeccf34dfac048c72b85$

Table 24. 15th round state value

State	State Recovery Value
S_8^{15R}	$0xfa4c806e6afb730ff2817bd58b1600fa$
S_7^{15R}	$0xe63abec1c749684d16d488a8b607e883$
S_6^{15R}	$0xb6601ff97f3d396f30d0f07f5b629e21$
S_5^{15R}	$0xe9bc47796839592b52a7ef871a70d403$
S_4^{15R}	$0xb57f888bdfb47314856358c2fe37d8fb$
S_3^{15R}	$0x3fbd6bf7b23084b2ab45c60f41a7b358$
S_2^{15R}	$0x5cf15fd35372d49383d72c96a9d79969$
S_1^{15R}	$0x20f1c1670c471c5835412dfb3d65c74a$

The forgery tag value is then verified by performing the decryption process. Decryption is carried out using the ciphertext input of the forgery target in Table 6, the forgery tag value $0xcb28ff49c79c5db4361064163f36aeaa$, the key, and the nonce. The decryption result confirms that the forgery tag is valid, and the plaintext is obtained as shown in Table 5. Thus, the forgery attack on PAES-8 was successful.

5. Analysis

This section describes the analysis of the results of the attack implementation. The factors observed include the suitability of Algorithm 5 and the required attack complexity. Based on the results of the the attack implementation, the following conclusions are drawn:

1) At the recovery stage, S_8recovery stage, the recovery of S_8^{0R} is performed by solving $2 \times 16 = 32$ differential equations using the 16 differential attack trajectories generated by the Cancellation $(\Delta P_\alpha, \Delta P_\beta)$ with a probability of 2^{-6}, and 16 differential attack trajectories generated by Cancellation with a probability $(\Delta P_\alpha, \Delta P_\beta)$ of 2^{-7}. This is because solving two differential equations using differential attack trajectories generated by cancellation probabilities 2^{-6} and 2^{-7} cannot determine a single exact solution. Thus, injection using ΔP_α and ΔP_β with cancellation probability 2^{-7} must also be performed. This demonstrates a mismatch with the attack provisions of Sasaki and Wang (2014), which states that injection using ΔP_α and ΔP_β with cancellation probability 2^{-7} should only be performed when there is no cancellation from injection using ΔP_α and ΔP_β with cancellation probability 2^{-6}. Algorithm 5 shows the forgery attack on PAES-8, which has been adapted to the implementation results.
2) The requirement to inject using ΔP_α and ΔP_β with cancellation probability 2^{-7} causes an increase in attack complexity by 16.2^8. Thus, the complexity of the forgery attack on PAES-8 becomes $16.2^7 + 16.2^8 + 2^7 = 2^{11} + 2^{12} + 2^7 \approx 2^{12}$.

Algorithm 5 Universal Forgery Attack Adjustment on PAES-8

Input : Plaintext $P = (P_0 \parallel P_1 \parallel \cdots \parallel P_{14})$
Output : Ciphertext $C = C_0 \parallel C_1 \parallel \cdots \parallel C_{14})$ and tags T

1. Query the first 15 plaintext blocks of the target $P = (P_0 \parallel P_1 \parallel \cdots \parallel P_{14})$ and get the key stream $R_0, R_1, \ldots, R_{14}$.
2. **for** $posisi = 1$ **to** 16 **do**
3. **for** $k = 1$ **to** 2^7 **do**
4. Select a different plaintext ΔP_α^k with 1 byte active at position and find ΔP_β^k that corresponds.
5. Query $(P_0 \oplus \Delta P_\alpha^k \parallel P_1 \oplus \Delta P_\beta^k \parallel P_2 \parallel \cdots \parallel P_{14})$ and get the key stream $R_0^k, \ldots, R_{14}^k$.
6. Check if the difference $R_7 \oplus R_7^k$ can produce $R_7 \oplus R_7^k \oplus R_8 \oplus R_8^k$ with AES_0.
7. Check the same properties on the next 4 rounds.
8. Keep the pairs that satisfy 6 and 7.
9. **end for**
10. **for** $l = 1$ **to** 2^8 **do**
11. Select a different plaintext ΔP_α^l with 1 byte active at position and find ΔP_β^l that corresponds.
12. Query $(P_0 \oplus \Delta P_\alpha^l \parallel P_1 \oplus \Delta P_\beta^l \parallel P_2 \parallel \cdots \parallel P_{14})$ and get the key stream $R_0^l, \ldots, R_{14}^l$.
13. Check if the difference $R_7 \oplus R_7^l$ can produce $R_7 \oplus R_7^l \oplus R_8 \oplus R_8^l$ with AES_0.
14. Check the same properties on the next 4 rounds.
15. Keep the pairs that satisfy 13 and 14.
16. **end for**
17. Restore the byte at the position of the state word S_8 on loop 0.
18. **end for**
19. Recover S_7 on rounds 8, 9, 10, 11, and 12.
20. **for** $k = 1$ **to** 2^7 **do**
21. Select the difference ΔP_α^k with 1 byte active at position and find ΔP_β^k that corresponds.
22. Query $(P_0 \parallel P_1 \parallel P_2 \oplus \Delta P_\alpha^k \parallel P_3 \oplus \Delta P_\beta^k \parallel \cdots \parallel P_{14})$ and get the key stream $R_0^k, \ldots, R_{14}^k$.
23. Check if the difference $R_9 \oplus R_9^k$ can produce $R_9 \oplus R_9^k \oplus R_{10} \oplus R_{10}^k$ with AES_0.
24. Check the same properties on the next 4 rounds.
25. Save the pairs that satisfy 23 and 24.
26. **end for**
27. Recover S_7 on laps 13 and 14.
28. Determine all state*s* in round 8.
29. Continue the rest of the transformation and generate the tag value.

6. Conclusion

This study successfully implemented and evaluated a forgery attack on the PAES-8 encryption scheme, based on the theoretical concept proposed by Sasaki and Wang. Our findings reveal that the attack cannot be executed without significant modifications, such as injecting differential plaintext pairs with high-probability outputs in the AES DDT. These modifications substantially increase the attack's complexity, indicating that PAES-8 may offer greater resistance to forgery attacks than initially anticipated.

From a practical perspective, this research provides valuable insights into the limitations of theoretical attacks and highlights the challenges of applying such attacks to real-world encryption schemes. These findings are significant for researchers and cryptography practitioners, emphasizing the need for more comprehensive methods to analyze the security of authenticated encryption schemes.

Furthermore, this study has practical implications for the future design of encryption schemes. The additional complexity identified suggests that encryption schemes should be rigorously tested beyond theoretical scenarios to ensure robustness in real-world environments. In the long term, these findings can guide the industry toward improving cryptographic designs to better withstand emerging threats.

However, the study also acknowledges its limitations. Our implementation focused on the nonce-misuse model, and further research is needed to evaluate PAES-8 under the nonce-respecting model to gain a more comprehensive understanding of the scheme's resilience. Additionally, similar attacks should be applied to other

authenticated encryption schemes to determine whether the additional complexity observed in this study is a common phenomenon in modern cryptography.

7. Acknowledgements

The authors would like to express their sincere gratitude to Politeknik Siber dan Sandi Negara for their invaluable support and resources, which were essential to the successful completing of this research. The guidance and facilities provided by the institution significantly enhanced the depth and quality of this study. We are also grateful for the collaborative environment and academic encouragement offered by the faculty and staff, which fostered the innovation and rigor required for this work.

References

Aumasson, J.-P., Jovanovic, P., & Neves, S. (2015). *NORX8 and NORX16: Authenticated Encryption for Low-End Systems*. https://eprint.iacr.org/2015/1154

Banik, S., Chakraborti, A., Iwata, T., Minematsu, K., Nandi, M., Peyrin, T., Sasaki, Y., Sim, S. M., & Todo, Y. (2021). *GIFT-COFB v1.1*. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/gift-cofb-spec-final.pdf

Bao, Z., Chakraborti, A., Datta, N., Guo, J., Nandi, M., Peyrin, T., & Yasuda, K. (2021). *PHOTON-Beetle Authenticated Encryption and Hash Family*. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/photon-beetle-spec-final.pdf

Bellare, M., & Namprempre, C. (2008). Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, *21*, 469–491. https://doi.org/10.1007/s00145-008-9026-x

Bernstein, D. (2013). *CAESAR: Call for Submissions*.

Bhaumik, R., & Nandi, M. (2017). *Improved Security for OCB3*. https://eprint.iacr.org/2017/845

Canteaut, A., Duval, S., Leurent, G., Naya-Plasencia, M., Perrin, L., Pornin, T., & Schrottenloher, A. (2020). Saturnin: a suite of lightweight symmetric algorithms for post-quantum security. *IACR Transactions on Symmetric Cryptology*, *2020*(Special Issue 1), 160–207. https://doi.org/10.13154/TOSC.V2020.IS1.160-207

Charlton, E. (2024). Cybersecurity: Rising Threats and System Safety. *World Economic Forum Agenda*. https://www.weforum.org/agenda/2024/01/cybersecurity-cybercrime-system-safety/

Choi, W., Hwang, S., Lee, B., & Lee, J. (2024). ZLR: a fast online authenticated encryption scheme achieving full security. *Designs, Codes and Cryptography*. https://doi.org/10.1007/s10623-024-01434-6

Daemen, J., Hoffert, S., Peeters, M., Van Assche, G., & Van Keer, R. (2020). Xoodyak, a lightweight cryptographic scheme. *IACR Transactions on Symmetric Cryptology*, *2020*(Special Issue 1), 60–87. https://doi.org/10.13154/TOSC.V2020.IS1.60-87

Dobraunig, C., Mendel, F., Eichlseder, M., & Schläffer, M. (2021). *Ascon v1.2 Submission to NIST* (p. 52). https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/ascon-spec-final.pdf

Jean, J., Nikolić, I., Peyrin, T., & Seurin, Y. (2021). The Deoxys AEAD Family. *Journal of Cryptology*, *34*. https://doi.org/10.1007/s00145-021-09397-w

Jean, J., Nikolic, I., Sasaki, Y., & Wang, L. (2016). Practical forgeries and distinguishers against PAES. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, *99*, 39–48. https://doi.org/10.1587/transfun.E99.A.39

Jimale, M. A., Z'aba, M. R., Kiah, M. L. B. M., Idris, M. Y. I., Jamil, N., Mohamad, M. S., & Rohmad, M. S. (2022). Authenticated Encryption Schemes: A Systematic Review. *IEEE Access*, *10*, 14739–14766. https://doi.org/10.1109/ACCESS.2022.3147201

Liu, F., & Liu, F. (2017). Universal Forgery with Birthday Paradox: Application to Blockcipher-based Message Authentication Codes and Authenticated Encryptions. *IACR Cryptology EPrint Archive*, *2017*, 653. https://eprint.iacr.org/2017/653.pdf

NIST. (2001). *Advanced Encryption Standard (AES)* (Issue FIPS PUB 197). https://csrc.nist.gov/pubs/fips/197/final

Sasaki, Y., & Wang, L. (2014). *A Practical Universal Forgery Attack against PAES-8* (p. 218). Citeseer. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.483.4356&rep=rep1&type=pdf

Schroé, W. (2015). *Cryptanalysis of Submission to the CAESAR Cryptographic Competition iFeed*. https://www.esat.kuleuven.be/cosic/publications/thesis-262.pdf

Statista. (2024). *Cost of cybercrime worldwide forecast*. https://www.statista.com/forecasts/1280009/cost-cybercrime-worldwide

Van Tilborg, H. C., & Jajodia, S. (Eds.). (2011). *Encyclopedia of Cryptography and Security* (2nd ed.). Springer Science+Business Media, LLC. https://doi.org/10.1007/978-1-4419-5906-5

Wu, H., & Preneel, B. (2013). *AEGIS: A Fast Authenticated Encryption Algorithm*. https://eprint.iacr.org/2013/695

Ye, D., Wang, P., Hu, L., Wang, L., Xie, Y., Sun, S., & Wang, P. (2014). *Parallelizable Authenticated Encryption Schemes based on AES Round Function*. http://competitions.cr.yp.to/round1/paesv1.pdf