<table>
<thead>
<tr>
<th>Title</th>
<th>Practical Forgeries and Distinguishers against PAES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Jean, Jérémý; Nikolic, Ivica; Sasaki, Yu; Wang, Lei</td>
</tr>
<tr>
<td>Date</td>
<td>2016</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/10220/39784">http://hdl.handle.net/10220/39784</a></td>
</tr>
<tr>
<td>Rights</td>
<td>© 2016 Institute of Electronics, Information and Communication Engineers. This paper was published in IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences and is made available as an electronic reprint (preprint) with permission of Institute of Electronics, Information and Communication Engineers. The published version is available at: [<a href="http://dx.doi.org/10.1587/transfun.E99.A.39">http://dx.doi.org/10.1587/transfun.E99.A.39</a>]. One print or electronic copy may be made for personal use only. Systematic or multiple reproduction, distribution to multiple locations via electronic or other means, duplication of any material in this paper for a fee or for commercial purposes, or modification of the content of the paper is prohibited and is subject to penalties under law.</td>
</tr>
</tbody>
</table>
Practical Forgeries and Distinguishers against PAES*

Jérémie JEAN(a), Ivica NIKOLIĆ(b), Nonmembers, Yu SASAKI(c), Member, and Lei WANG(d), Nonmember

SUMMARY We present two practical attacks on the CAESAR candidate PAES. The first attack is a universal forgery for any plaintext with at least 240 bytes. It works for the nonce-repeating variant of PAES and in a nutshell it is a state recovery based on solving differential equations for the S-Box leaked through the ciphertext that arise when the plaintext has a certain difference. We show that to produce the forgery based on this method the attacker needs only \(2^{11}\) time and data. The second attack is a distinguisher for \(2^{64}\) out of \(2^{128}\) keys that requires negligible complexity and only one pair of known plaintext-ciphertext. The attack is based on the lack of constants in the initialization of the PAES which allows to exploit the symmetric properties of the keyless AES round. Both of our attacks contradict the security goals of PAES.

key words: PAES, universal forgery, distinguisher, symmetric property, authenticated encryption

1. Introduction

The CAESAR competition [2] (Competition for Authenticated Encryption: Security, Applicability, and Robustness) has started in March 2014, and its goal is to improve the understanding of the crypto community in the area of authenticated ciphers through a public competition for submitting authenticated encryption schemes that offer advantages over the widely used AES–GCM [3]. In total, 57 ciphers were submitted to the open call, and in the following three years, through security analysis and investigation of the implementations advantages, it is expected that among these ciphers, a few to be selected in a portfolio of recommended authenticated schemes that are suitable for widespread adoption.

A number of the proposed CAESAR candidates (as well as the benchmark AES–GCM) are based on the current encryption standard: the AES family of block ciphers. The reason for this is twofold. First, the AES has undergone an extensive analysis and is assumed that its security is well understood (or at least better understood compared to all of the remaining unbroken ciphers). Second, AES offers a large software implementation advantage on the latest processor through the so-called AES–NI instruction set, i.e., modern processors have dedicated instructions that allow to reduce the execution time of the AES cipher calls.

In general, the CAESAR candidates based on the AES use the block cipher in two ways: either as a whole (or a variant consisting of at least a certain number of rounds), or only its round function. The first type of candidates (OCB [4], AES–COPA [5], etc., and AES–GCM) are constructions that require calls to the full 10-round AES–128 (or at least 4-round variants with independent round keys). Usually, they are provable modes based on security reduction to the security of AES, and thus benefit from the current state-of-the-art cryptanalysis of AES–128 [6]. The second type uses only the AES round function and has no strict security proof, i.e., the mode is not provably secure, however, the resistance against common attacks is provided through ad-hoc techniques. Such candidates (see AEGIS [7], PAES [8], Tiaoxin–346 [9]) benefit from the good security properties and the software performance of the AES round function. They tend to use less than 10 AES round calls per message blocks, and as such are extremely fast.

1.1 Our Contributions

We provide a cryptanalysis of the CAESAR candidate PAES [8] and show two attacks that contradict the security claims given by the designers. Common for both of the attacks are the low complexity requirements and the misuse of the AES round function in PAES.

The first attack targets the nonce-repeating mode of PAES (called PAES–8) and is a universal forgery attack of any plaintext with at least 240 bytes. It requires \(2^{11}\) time and data complexity to fully recover the internal state and to produce forgery. To launch the attack, we use a special differential trail that can take two different paths. By analyzing the ciphertext difference, the path is uniquely determined and this leads a state recovery based on the differential property of the AES S-Box. Our attack shows that a mere differential analysis (often given by providing the best differential characteristic of a construction) is insufficient for proving security in the nonce-repeating mode, even when the candidates guarantees multiple applications of AES round function.

The second attack comes in a form of a distinguisher for a class of \(2^{64}\) weak keys among the total \(2^{128}\) keys of PAES. We show that if the attacker can control the nonce, then a single pair of known plaintext and corresponding ciphertext is sufficient to distinguish PAES from an ideal au-
The design resembles a stream cipher: it has an initialization phase, which will be used to compute the ciphertext. Finally, the state is mixed with the associated data length $A_{\text{data}}$ and and the message length $M_{\text{len}}$ as the initialization process, and 128-bit tag $T$ is produced from a part of the state. The computation structure is illustrated in Fig. 1. In the paper, for simplicity, we assume that the associated data is always set to empty. Note that our attacks on PAES-8 work equally well when the associated data is not an non-empty string (but our distinguisher on PAES-4 requires the associated data to be empty).

The encryption takes as input a variable-length plaintext, a 128-bit key, a 128-bit nonce and produces a variable-length ciphertext and a 128-bit authentication tag. The decryption takes as input a variable-length ciphertext, a 128-bit key, a 128-bit nonce and a 128-bit authentication tag and produces a variable-length plaintext and a 128-bit authentication tag. If the computed tag matches the received tag, it outputs the plaintext. Otherwise, it outputs decryption failure symbol $. The difference between PAES-4 and PAES-8 lies in the size of the internal state, which amounts to four 128-bit blocks for the former, and eight 128-bit blocks for the latter. A functional difference between these two variants is in the mode: PAES-4 has security claims only in the nonce-respecting mode, while PAES-8 in both, the nonce-respecting and nonce-repeating modes.

To simplify the presentation, in the sequel we describe only PAES-8, and only as authenticated encryption. The design resembles a stream cipher: it has an initialization (where the key and the nonce are loaded into the state), then it processes the input message and produces the ciphertext, and finally in the finalization it produces the tag. The internal state $S$ has eight words $S_1, S_2, \ldots, S_8$, each of 128 bits, i.e., $|S_i| = 128, i = 1, \ldots, 8$. The state update function $\text{StateUpdate}(S, M)$ is the round transformation and uses eight keyless$^1$ AES-round calls (denoted further as $\text{AES}_{0}$) to update the state as depicted in Fig. 2.

### 2.1 Initialization

The 128-bit master key $K$ and the nonce $N$ are loaded into the eight words of the state, the state goes through 10 rounds and at the end the key is XORed to all eight words of the state:

$$S_1 = K \oplus N, \quad S_5 = L^4(K) \oplus L^7(N)$$
$$S_2 = L(K) \oplus L^2(N), \quad S_6 = L^6(K) \oplus L^8(N)$$
$$S_3 = L^2(K) \oplus L^8(N), \quad S_7 = L^6(K) \oplus L^5(N)$$
$$S_4 = L^3(K) \oplus L^5(N), \quad S_8 = L^7(K) \oplus L^6(N)$$

for $i = 1$ to $10$

\footnote{We emphasize that all the $\text{AES}$ calls are keyless, that is, composed of SubBytes, ShiftRows and MixColumns (but no AddRoundKey).}

<table>
<thead>
<tr>
<th>Design</th>
<th>Supported nonce modes</th>
<th>Attack</th>
<th>Attack mode</th>
<th>Size of key class (out of $2^{128}$)</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAES-4</td>
<td>respecting</td>
<td>distinguisher</td>
<td>respecting</td>
<td>$2^{64}$</td>
<td>1</td>
</tr>
<tr>
<td>PAES-8</td>
<td>respecting+repeating</td>
<td>universal forgery</td>
<td>repeating</td>
<td>$2^{128}$</td>
<td>$2^{11}$</td>
</tr>
<tr>
<td>PAES-8</td>
<td>respecting+repeating</td>
<td>distinguisher</td>
<td>respecting+repeating</td>
<td>$2^{64}$</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 1** Attacks on PAES.
$S = \text{StateUpdate} (\text{State}, 0)$

$\text{for } i = 1 \text{ to } 8$

$S_i = S_i \oplus K$

where $L$ is a linear transformation that operates on the four 32-bit columns $a, b, c, d$ of a 128-bit word $a||b||c||d$, and is defined as $L(a, b, c, d) = (b, c, d \oplus a, a)$. With $L^i$ we denote the $i$-th functional power of the transformation $L$, e.g., $L^2 = L \circ L$.

### 2.2 Processing the Plaintext

In one round, from 16-byte plaintext $P_i$, 16-byte ciphertext $C_i$ is obtained with one call to the \text{StateUpdate} function (see Fig. 3):

\[
\text{tmp} = S_7 \\
\text{StateUpdate} (S, P_i) \\
R_i = \text{tmp} \oplus S_7 \\
C_i = P_i \oplus R_i
\]

The decryption can be defined accordingly (see Fig. 4), where all the state but for $S_8$ are updated and the plaintext $P_i$ is recovered as $R_i \oplus C_i$ and then the state $S_8$ is updated. The initialization, finalization and the tag production is the same as the encryption process. Note that so called Releasing Unverified Plaintext (RUP) is not defined for the decryption. Thus, the recovered $P$ are output only if the tag authentication is successful. Otherwise, the decryption failure symbol $\perp$ is output.

### 2.3 Finalization and the Tag Production

Let $M_{len}$ be the 128-bit encoding of the message length. Then, the tag $T$ is produced after 10 rounds of the \text{StateUpdate} function where the message input is set to $M_{len}$:

\[
\text{for } i = 1 \text{ to } 10 \\
\text{StateUpdate} (S, M_{len}) \\
T = S_7 \oplus S_8
\]

### 2.4 Claimed Security of PAES

The claimed security of PAES is given in Table 2. We emphasize in particular that 128-bit security is claimed for the integrity of PAES in the nonce-repeating mode.

#### 3. Practical Universal Forgery against PAES-8

In this section, we show a universal forgery attack for PAES-8 in the nonce-repeating mode. The attack works for any plaintext with length of at least 240 bytes, and requires only a small time and data complexity. The steps of the attack can be summarized as follows:

1. Inject differences in two consecutive plaintext blocks such that they cancel in $S_8$ with a high probability.
2. The ciphertext difference after eight rounds will reveal if the cancellation in $S_8$ occurred and if so, it will leak information about the state bits.
3. Once the state is recovered, the tag is produced by going through the remaining of the transformations of the (now) public construction.

#### 3.1 Differential Trail and Detection of Difference Cancellation

The differential trail used in the attack is given in Fig. 5. We inject difference $\Delta \alpha$ in the plaintext $P_0$, and try to cancel it with another difference $\Delta \beta$ in the plaintext $P_1$.

Interestingly, this type of trail has been discussed by the designers of PAES (see [8, Figure 4.3]), however, they focused on the standard case of propagating the difference through eight rounds and tried to predict it. On the other hand, we use a different approach: our goal is not to predict the difference after eight rounds, but only to detect if the initial differences in $\Delta \alpha$ and $\Delta \beta$ have canceled.

By injecting the difference $\Delta \alpha$ and $\Delta \beta$ according to Fig. 5, the differential trail can take two patterns:

1. The differences $\Delta \alpha$ and $\Delta \beta$ cancel, thus only the words

<table>
<thead>
<tr>
<th>Goal</th>
<th>Nonce-respecting</th>
<th>Nonce-repeating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidentiality for the plaintext</td>
<td>128</td>
<td>-</td>
</tr>
<tr>
<td>Integrity for the plaintext</td>
<td>128</td>
<td>-</td>
</tr>
<tr>
<td>Integrity for the associated data</td>
<td>128</td>
<td>-</td>
</tr>
<tr>
<td>Integrity for the public message number</td>
<td>128</td>
<td>-</td>
</tr>
</tbody>
</table>
Fig. 5 Differential trail used in the attack. The bold lines denote active state words.

2. The differences $\Delta \alpha$ and $\Delta \beta$ do not cancel and there are additional active words. The trail for this case is shown in Fig. 6.

Both of the differential trails in Fig. 5 and Fig. 6, the difference appears from $\Delta R_7$. This makes the analysis non-trivial to detect the occurrence of the cancellation between $\Delta \alpha$ and $\Delta \beta$. In the following section, we explain how to detect the cancellation between $\Delta \alpha$ and $\Delta \beta$. We further show the optimal choices of $\Delta \alpha$ and $\Delta \beta$.

3.1.1 Choosing Plaintext Differences $\Delta \alpha$ and $\Delta \beta$

For an arbitrary difference $\Delta \alpha$ in the plaintext $P_0$, the difference $\Delta \beta$ in the plaintext $P_1$ should be chosen such that it will cancel $\Delta \alpha$ and thus will avoid activating the state $S_8$. Therefore, $\Delta \alpha$ and $\Delta \beta$ are chosen so that the cancellation can occur with a high probability – this happens when $\Delta \alpha$ has only one active byte. Let $\alpha$ and $\beta$ be the input and output difference transition of the S-Box, i.e., $\alpha$ changes to $\beta$ with a probability $2^{-6}$. Then, $\Delta \alpha$ and $\Delta \beta$ are defined as

$$\Delta \alpha = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),$$
$$\Delta \beta = \text{MixColumns} \circ \text{ShiftRows} (\beta, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),$$

and thus $\Delta \alpha$ after AES will change to $\Delta \beta$ with probability $2^{-6}$. We note that the difference $\alpha$ can be located in any of the 16 bytes of the state. The above analysis is depicted in Fig. 7.

3.1.2 Detecting the Cancellation between $\Delta \alpha$ and $\Delta \beta$

We can detect if the cancellation occurred by observing the differences in the ciphertexts $C_i$ (or equivalently, the difference in the key streams $R_i$) after eight rounds. There are two possible cases:

- **Cancellation occurred.** From the trail on Fig. 5, it follows that the difference $\Delta R_8 \oplus \Delta R_7$ is obtained when $\Delta R_7$ goes through one AES round. (The focused AES round is stressed by dotted circle in Fig. 5.) It means that the difference in each of the 16 bytes of $\Delta R_7$ can produce the corresponding differences in the bytes of $\text{ShiftRows}^{-1} \circ \text{MixColumns}^{-1} (\Delta R_8 \oplus \Delta R_7)$ through the S-Box. We note that the probability of this event is one when the cancellation occurred.

- **Cancellation did not occur.** If the cancellation did not occur, then there are additional state words with differences (marked with “$\Delta \neq 0$” in Fig. 6). In this case, $\Delta R_8 \oplus \Delta R_7$ is obtained when $\Delta R_7 \oplus \Delta X$ (where $\Delta X$ is the non-zero difference in $S_6$) goes through AES. In contrast to the above case, now $\Delta R_7$ may not be able to produce $\text{ShiftRows}^{-1} \circ \text{MixColumns}^{-1} (\Delta R_8 \oplus \Delta R_7)$ through the S-Box. The gap of the probability of this
equations of the form $S_{all 16 \text{ bytes}}$ are active is $\approx 2^{-2}$. Without loss of generality, we can assume that $\Delta x$ is active in all 16 bytes. Therefore, when $\Delta x$ and $\Delta y$ canceled each other, the probability of a 16-byte match is 1, however, when they do not cancel each other, then the probability drops to $2^{-16}$. As a result, we can easily distinguish the above two cases, by analyzing $\Delta R_7$ and $\Delta R_8$.

The same distinguishing method can be applied to 4 additional rounds (see Fig. 8). This way, we can increase the probability of distinguishing the two cases, and end up with a very low probability of matching differences through S-Boxes in the case when $\Delta x$ and $\Delta y$ do not cancel. As we apply it to five rounds, the probability becomes $2^{-5 \cdot 16} = 2^{-80}$.

3.2 Recovery of State Words

Assume that $\Delta x$ and $\Delta y$ have canceled (as demonstrated above, we can single out the case when they cancel). It means that we have the input difference $\Delta R_7$ and the output difference $\Delta R_8 \oplus \Delta R_7$ of an active AES$_0$ for the word $S_7$, i.e.

$$
\text{SubBytes}(R_7) \oplus \text{SubBytes}(R_7 \oplus \Delta R_7) = \text{ShiftRows}^{-1} \circ \text{MixColumns}^{-1}(\Delta R_8 \oplus \Delta R_7).
$$

As in $S_7$, all 16 bytes are active (with a probability very close to 1), we can easily find the values of the individual bytes by the well-known method of solving 16 differential equations of the form $S(x \oplus \Delta \text{input}) \oplus S(x) = \Delta \text{output}$ that come from the system using S-Box $S$. Each such equation on average has two solutions, because if $x$ is a solution, then $x \oplus \Delta \text{input}$ is also a solution. To find a single solution for each byte, we repeat once the recovery for different $\Delta x$ and $\Delta y$.

As a result, we can recover the value of $S_7$ at round 8 of the encryption.

Using the very same method, we can recover $S_7$ at rounds 9, 10, 11 and 12. For instance, for round 9, the input (resp. output) difference of AES$_0$ is $\Delta R_7 \oplus \Delta R_8$ (resp. $\Delta R_7 \oplus \Delta R_8 \oplus \Delta R_9$). With the knowledge of the values of 5 consecutive $S_7$, we can uniquely recover the values of $S_6, S_5, S_4, S_3$ at round 8 by simple computation using those words.

We can recover two more $S_7$ words (of additional 2 rounds) if we shift the round where we apply the difference $\Delta x$. Namely, we introduce $\Delta x$ at $P_2$ instead of $P_0$ and introduce $\Delta y$ at $P_3$. Hence, we will have the values of $S_7$ for 7 consecutive rounds.

The state word $S_8$ is different compared to the remaining seven words and it is not possible to recover it by using the above method. Nevertheless, we can still recover $S_8$ at round 0 of the encryption based on the differences $\Delta x$ and $\Delta y$, i.e., we can recover the active byte where the difference $\Delta x$ is non-zero. By repeating the recovery with 16 different positions of active bytes, we can deduce the whole state word $S_8$ at round 0. As $S_8$ does not take feedback from any other word (but the plaintext), we can easily find the value of $S_8$ at any round, including our target round 8. That is, with the knowledge of $S_7$ of seven consecutive rounds (8, 9, ..., 14) which can be deduced as shown above, and $S_8$ at round 8, we can recover the full state at round 8.

3.3 Attack Procedure

We now present the universal forgery attack. The goal of the attack is to produce a tag of an arbitrary plaintext. In our case, the attack works as long as the length of the plaintext is at least 16 blocks (240 bytes). Our forgery is based on a state recovery, i.e., if at some round the whole state is known, then the tag can easily be produced by performing the remaining operations of the finalization, and therefore it can be produced offline.

Let $P_0, P_1, \ldots, P_{14}$ be the first 15 blocks of the plaintext. Then, the forgery can be described with the following Algorithm 1.

The first loop is used to recover $S_8$, and to recover five $S_7$, and the second to recover the remaining two $S_7$. Note, each of the loops (the inner loop of the first loop) will produce two pairs, as the probability of the trail in the top ($\Delta x$ will be canceled by $\Delta y$) is $2^{-6}$. In case no good trails with probability $2^{-6}$ exist, the attacker can switch to ones with probability $2^{-7}$ and run the loops $2^8$ times. Furthermore, as we have seen from the previous analysis, a probability of false positives is very low (around $2^{-80}$).

From the algorithm, it follows that the time complexity of the attack is $16 \cdot 2^7 + 2^7 \approx 2^{11}$ computations. The data complexity is similar and comes in a form of chosen plaintexts. To solve efficiently the differential equations, the attack needs about $2^{16}$ bytes in memory.
Algorithm 1: Universal forgery attack

1: Query the first 15 plaintext blocks of the target (P_0||P_1||...||P_{14}), and obtain the key stream R_0, R_1, ..., R_{14}.
2: for position = 1 to 16 do
3:   for i = 1 to 2^7 do
4:     Choose 1-byte difference Δ̄_i with active byte at position and find the corresponding Δρ̄.
5:     Query (P_0||Δ̄_i||P_1||Δρ̄||P_2||...||P_{14}) and obtain the key stream R_0, ..., R_{14}.
6:     Check if the difference R_7 ⊕ R_6 can result in R_7 ⊕ R_6 ⊕ R_8 ⊕ R_9 by AES_0.
7:     Check the same property for next 4 additional rounds.
8:   Save the pairs that pass all the above checks.
9: end for
10: Recover the byte at position of the state word S_8 at round 0.
11: end for
12: Recover S_7 at rounds 8,9,10,11,12.
13: for i = 1 to 2^7 do
14:   Choose 1-byte difference Δ̄_i and find the corresponding Δρ̄.
15: Query (P_0||Δ̄_i||P_1||Δρ̄||P_2||...||P_{14}) and obtain the key stream R_0, ..., R_{14}.
16: Check if the difference R_0 ⊕ R_6 can result in R_0 ⊕ R_6 ⊕ R_10 ⊕ R_16 by AES_0.
17: Check the same property for next 4 additional rounds.
18: Save the pairs that pass all the above checks.
19: end for
20: Recover S_7 at rounds 13 and 14.
21: Deduce all the state words at round 8.
22: Go through the remaining of the transformations and produce the tag.

4. Practical Distinguisher for a Weak-Key Class of PAES-4 and PAES-8

We continue our analysis by presenting a distinguisher for a class of 2^64 weak keys (out of 2^{128} keys) in PAES-8. The distinguisher requires negligible time complexity and only a single pair of known plaintext-ciphertext and a chosen nonce. It exploits the lack of constants in the design and the symmetric properties of the keyless AES round function. We give a thorough description of the distinguisher for the nonce-respecting mode of PAES-8, as well as a brief description of a similar distinguisher for the nonce-respecting mode PAES-4.

4.1 Symmetric Properties of the AES Round Function

We first recall the known symmetric property of the AES round function [11]. Namely, if a state is symmetric in the sense that its two halves are equal, then the keyless round function AES_0 of the AES maintains this property. We recall the property of [11] using block matrices, and we introduce the following more general notations:

\[ U(A, B) = \begin{pmatrix} A & A \\ B & B \end{pmatrix}, \]
\[ V(A, B) = \begin{pmatrix} A & B \\ B & A \end{pmatrix}, \]
\[ W(A, B) = \begin{pmatrix} A & B \\ A & B \end{pmatrix}. \]

Additionally, we denote by U, V and W the associated sets respectively for all possible values of the 2x2 block matrices A and B. Finally, we denote M the constant MDS matrix used in the AES round function, and observe that:

\[
M = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}
= \begin{pmatrix} M_1 & M_2 \\ M_2 & M_1 \end{pmatrix} = V(M_1, M_2) \in V.
\]

Property 1. Let S ∈ U. Then, AES_0(S) ∈ U.

Proof. Let S = U(A, B) ∈ U, and write the bytes in S as:

\[
\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} x_0 & x_4 \\ x_1 & x_5 \\ x_2 & x_6 \\ x_3 & x_7 \end{pmatrix} \]

As the \textbf{SubBytes} operation applies the same bijection to all the bytes in the state, we ignore it here as it obviously preserves the structure. After the \textbf{ShiftRows} operation, the state becomes

\[
\begin{pmatrix} x_0 & x_4 \\ x_5 & x_1 \\ x_2 & x_6 \\ x_3 & x_7 \end{pmatrix} \defeq \begin{pmatrix} A' \\ B' \end{pmatrix},
\]

thus it still belongs to U. Then, the \textbf{MixColumns} operation results in:

\[
\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \times \begin{pmatrix} x_0 & x_4 \\ x_5 & x_1 \\ x_2 & x_6 \\ x_7 & x_3 \end{pmatrix} = \begin{pmatrix} M_1 & M_2 \\ M_2 & M_1 \end{pmatrix} \times \begin{pmatrix} A' \\ B' \end{pmatrix}
= \begin{pmatrix} M_1A' \oplus M_2B' \\ M_2A' \oplus M_1B' \end{pmatrix}
\defeq \begin{pmatrix} A'' \\ B'' \end{pmatrix} \in U.
\]

\[ \square \]

Property 2. Let S ∈ W. Then, AES_0(S) ∈ V, and AES_0(AES_0(S)) ∈ W.

Proof. Let S = W(A, B) ∈ W, and write the bytes in S as:

\[
\begin{pmatrix} A \\ B \end{pmatrix} = \begin{pmatrix} x_0 & x_2 \\ x_1 & x_3 \\ x_4 & x_6 \\ x_5 & x_7 \end{pmatrix}
\]
Again, we ignore the SubBytes operation as the applied bijection preserves the structure of the internal states. However, after the ShiftRows operation the state becomes:

\[
\begin{pmatrix}
  x_0 & x_2 \\
  x_3 & x_5 \\
  x_6 & x_1 \\
  x_7 & x_1
\end{pmatrix}
\begin{pmatrix}
  x_4 & x_6 \\
  x_7 & x_1 \\
  x_0 & x_2 \\
  x_3 & x_5
\end{pmatrix}
= \begin{pmatrix}
  A' & B' \\
  B' & A'
\end{pmatrix} \in \mathcal{V},
\]

which is transformed by the subsequent MixColumns transformation into the state:

\[
\begin{pmatrix}
  M_1 & M_2 \\
  M_2 & M_1
\end{pmatrix}
\begin{pmatrix}
  A' & B' \\
  B' & A'
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  M_1A' + M_2B' & M_1B' + M_2A' \\
  M_2A' + M_1B' & M_2B' + M_1A'
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  A'' & B'' \\
  B'' & A''
\end{pmatrix} \in \mathcal{V}.
\]

Let \( X \rightarrow X' \) denote that the state \( X \) changes to \( X' \) by the ShiftRows operation. After applying a second keyless AES round, we get:

\[
\begin{pmatrix}
  A'' & B'' \\
  B'' & A''
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  y_0 & y_2 & y_4 & y_6 \\
  y_1 & y_3 & y_5 & y_7 \\
  y_4 & y_6 & y_0 & y_2 \\
  y_5 & y_7 & y_1 & y_3
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  y_0 & y_2 & y_4 & y_6 \\
  y_1 & y_3 & y_5 & y_7 \\
  y_4 & y_6 & y_0 & y_2 \\
  y_5 & y_7 & y_1 & y_3
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  y_0 & y_2 & y_4 & y_6 \\
  y_1 & y_3 & y_5 & y_7 \\
  y_4 & y_6 & y_0 & y_2 \\
  y_5 & y_7 & y_1 & y_3
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  A''' & B''' \\
  B''' & A'''
\end{pmatrix} \in \mathcal{W},
\]

and by the MixColumns:

\[
\begin{pmatrix}
  M_1 & M_2 \\
  M_2 & M_1
\end{pmatrix}
\begin{pmatrix}
  A''' & B''' \\
  B''' & A'''
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  M_1A''' + M_2B''' & M_1B''' + M_2A''' \\
  M_2A''' + M_1B''' & M_2B''' + M_1A'''
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  A'''' & B'''' \\
  B'''' & A''''
\end{pmatrix} \in \mathcal{W},
\]

which concludes the proof.

Finally, we can represent the action of the keyless AES rounds on the three sets \( \mathcal{U}, \mathcal{V} \) and \( \mathcal{W} \) as follows on Fig. 9.

![Fig. 9](image)

**Fig. 9** Action of AES\(_0\) of the symmetrical states from \( \mathcal{U}, \mathcal{V} \) and \( \mathcal{W} \).

4.2 Symmetric Properties of the PAES Transformations

Along with AES\(_0\), PAES uses a few more transformations, in particular, the XOR and the linear transformation \( L \). We investigate here how these two transformations preserve the class belongings.

**Property 3.** Let \( X \) be either \( \mathcal{U}, \mathcal{V} \) or \( \mathcal{W} \), and let \( S_1, S_2 \in X \). Then, \( S_1 \oplus S_2 \in X \).

**Proof.** Let \( S_1 = U(A_1, B_1), S_2 = U(A_2, B_2) \in \mathcal{U} \). Then:

\[
S_1 \oplus S_2 = \begin{pmatrix}
  A_1 & A_1 \\
  B_1 & B_1
\end{pmatrix} \oplus \begin{pmatrix}
  A_2 & A_2 \\
  B_2 & B_2
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  A_1 \oplus A_2 & A_1 \oplus A_2 \\
  B_1 \oplus B_2 & B_1 \oplus B_2
\end{pmatrix} \in \mathcal{U}.
\]

Let \( S_1 = V(A_1, B_1), S_2 = V(A_2, B_2) \in \mathcal{V} \). Then:

\[
S_1 \oplus S_2 = \begin{pmatrix}
  A_1 & A_1 \\
  B_1 & B_1
\end{pmatrix} \oplus \begin{pmatrix}
  A_2 & A_2 \\
  B_2 & B_2
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  A_1 \oplus A_2 & A_1 \oplus A_2 \\
  B_1 \oplus B_2 & B_1 \oplus B_2
\end{pmatrix} \in \mathcal{V}.
\]

Let \( S_1 = W(A_1, B_1), S_2 = W(A_2, B_2) \in \mathcal{W} \). Then:

\[
S_1 \oplus S_2 = \begin{pmatrix}
  A_1 & A_1 \\
  A_1 & A_1
\end{pmatrix} \oplus \begin{pmatrix}
  A_2 & A_2 \\
  A_2 & A_2
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  A_1 \oplus A_2 & A_1 \oplus A_2 \\
  A_1 \oplus A_2 & A_1 \oplus A_2
\end{pmatrix} \in \mathcal{W}.
\]

\[
\square
\]

**Property 4.** Let \( S \in \mathcal{W} \). Then, \( L(S) \in \mathcal{W} \).

**Proof.** Let \( S = W(A, B) \in \mathcal{W} \), and write the bytes in \( S \) as:

\[
S = \begin{pmatrix}
  A & B \\
  A & B
\end{pmatrix} = \begin{pmatrix}
  x_0 & x_2 & x_4 & x_6 \\
  x_1 & x_3 & x_5 & x_7
\end{pmatrix}
\]

Then:

\[
L(S) = \begin{pmatrix}
  x_0 & x_2 & x_4 & x_6 \\
  x_1 & x_3 & x_5 & x_7
\end{pmatrix}
\]

\[
= \begin{pmatrix}
  A_2 & A_0 \\
  A_3 & A_1
\end{pmatrix} \in \mathcal{W}.
\]

\[
\square
\]

4.3 The Distinguisher

To distinguish PAES, we use the first ciphertext \( C_0 \) produced
The state words $C$ and the nonce \( K \) are expanded with linear function \( L \) and 8 state words are computed by their linear combinations. Then, state update function is applied 10 times and finally \( K \) is XORed with all of the 8 state words. We first inspect how the state words \( S_1, S_2, \ldots, S_8 \) change the class belongings (either \( W \) or \( V \)) from the very first to the last steps of the initialization phase:

- \( K, N \in W \). By Properties 3 and 4, \( S_1, S_2, \ldots, S_8 \in W \) after the initial assignments in the initialization.
- After the first update. By Property 3, the XORs do not change the class belongings, thus each \( S_6, S_7, S_8 \) stay in \( W \) after the XORs at the top of the \textit{StateChanged}. Further, according to the Property 2, AES0 changes the class from \( W \) to \( V \). Consequently, at the end of the first update, \( S_i \in V, i = 1, \ldots, 8 \).
- The second update is similar to the previous one, but this time the class of \( S_i \) changes to \( W \).
- 
- After the tenth update. The classes of all \( S_i \) are \( W \).
- After the XORs of the key. As each \( S_i \) is in \( W \) and the key is in \( W \), by Property 3, it follows that each \( S_i \) will be in \( W \).

The propagation of the property during the initialization is described in Fig. 10.

![Property propagation in the initialization step.](image)

We now focus on the production of the ciphertext \( C_0 \). After the initialization all the state words belong to the class \( W \), thus obviously \( tmp = S_7 = W(A_1, B_1) \in W \). Because \( S_6, S_7 \in W \), \( S_6 \oplus S_7 \) also belongs to the class \( W \) due to Property 3. After the application of the \textit{StateChanged}, \( S_7 = V(A_2, B_2) \in V \) by Property 2. Thus, from the definition of the ciphertext \( C_0 = P_0 \oplus tmp \oplus S_7 \), we get:

\[
C_0 \oplus P_0 = \begin{pmatrix} A_1 \\ A_1 \end{pmatrix} \begin{pmatrix} B_1 \\ B_1 \end{pmatrix} \oplus \begin{pmatrix} A_2 \\ A_2 \end{pmatrix} \begin{pmatrix} B_2 \\ B_2 \end{pmatrix} = \begin{pmatrix} A_1 \oplus A_2 \\ A_1 \oplus A_2 \end{pmatrix} \begin{pmatrix} B_1 \oplus B_2 \\ B_1 \oplus B_2 \end{pmatrix} = \begin{pmatrix} X \\ Y \end{pmatrix} \begin{pmatrix} Z \\ T \end{pmatrix}.
\]

The propagation of the property during the production of \( C_0 \) is described in Fig. 11.

As shown in Eq. (1), only by looking the appearance of the matrix, there is no method to detect that the matrix is composed of two matrices; one belongs to the class \( W \) and the other belongs to the class \( V \). However, we can still apply some computation to distinguish the non-random behavior. Namely \( X \oplus Y \oplus Z \oplus T = 0 \), hence the XOR of the four 32-bit blocks of the first ciphertext and plaintext must result in a zero block. Therefore, we have a distinguisher which requires negligible complexity and only a single block of plaintext/ciphertext to distinguish PAES when instantiated with any of the \( 2^{64} \) keys and nonces from the class \( W \). We note that our computer simulation confirmed the correctness of the distinguisher.

Algorithm for Distinguisher

The goal of the distinguisher is determining if the interacting oracle is PAES or ideal primitive (authenticated encryption). The output of the distinguisher is determining bit \( b \in \{0, 1\} \), where \( b = 1 \) suggests that the oracle is PAES and \( b = 0 \) suggests that the oracle is the ideal primitive. The above distinguisher can be described in an algorithmic form as Algorithm 2.

**Algorithm 2: Distinguisher for weak keys**

**Input:** a weak key \( K \in W \)

**Output:** \( b \in \{0, 1\} \)

1: Choose any \( N \) satisfying the form \( W \).
2: Choose any 1-block message \( P_0 \).
3: Make a single query of \( (N, P_0) \) to the encryption oracle and obtain the corresponding ciphertext \( C_0 \).
4: Compute \( P_0 \oplus C_0 \) and as in Eq. (1), divide it into four sectors \( (X, Y, Z, W) \).
5: Compute \( tmp \leftarrow X \oplus Y \oplus Z \oplus W \).
6: if \( tmp = 0 \) then
7: return \( b = 1 \). //The oracle is PAES.
8: else
9: return \( b = 0 \). //The oracle is ideal primitive.
10: end if
4.4 Distinguisher for PAES-4

A similar distinguisher can be applied to PAES-4, in which the internal state size is composed of four words. A brief description of PAES-4 is given further.

In the initialization phase of PAES-4, the four state words \((S_1, S_2, S_3, S_4)\) are set exactly the same as the first four words of PAES-8. Then, the state is updated five times with the state update function of PAES-4, and finally the key \(K\) is XORed to each of the state words. The state update function for PAES-4 is described in Fig. 12.

We start the analysis by setting \(K, N \in \mathcal{W}\), which, after the expansion with \(L\) and the XOR, makes \(S_1, S_2, S_3, S_4 \in \mathcal{V}\). Then, the state update function is applied five times, which makes \(S_1, S_2, S_3, S_4 \in \mathcal{V}\). As a result, we end up with a different result than for PAES-8, in which after 10 rounds all state words were in the class \(\mathcal{W}\). In fact, this can be a problem for the distinguisher on PAES-4 as after the subsequent XOR of the key \(K \in \mathcal{W}\), the state words do not belong to a particular class. Nevertheless, as long as the associated data is empty, we can still apply the distinguisher.

After the initialization, the two middle state words are \(S_2 \oplus K\) and \(S_3 \oplus K\), where \(S_2, S_3 \in \mathcal{V}\) and \(K \in \mathcal{W}\). Obviously \(tmp = S_3 \oplus K = W(A_1, B_1) \oplus V(A_2, B_2)\). Furthermore, the XOR of two middle state words (just before the application of the AES round to the third word as shown in Fig. 12) results in \(S_2 \oplus K \oplus S_3 \oplus K = S_2 \oplus S_3 \in \mathcal{V}\), thus after the application of the AES round function, this updated word \(S_{new}^3\) belongs to \(\mathcal{W}\). Finally, the key stream can be represented as \(S_{new}^3 \oplus tmp = W(A'_1, B'_1) \oplus V(A_2, B_2)\), which yields the same distinguisher as for PAES-8.

5. Concluding Remarks

We have shown two practical attacks on the CAESAR candidate PAES: a universal forgery attack and a distinguisher, which contradict the security claims of this authenticated encryption scheme.

Our analysis gives insights into possible misuses of the AES round function. Although this transformation provides excellent resistance against differential and linear attacks (once it has been iterated several times), by no means it is a sufficient proof of security against all attacks. The designs based on the round function that does not apply any constants, as we have seen on the example of our distinguisher and the chosen-key rotational distinguisher [12] of PAES, are susceptible to attacks that exploit the symmetry of the AES transformations. Consequently, using random constants in such designs should be taken as a requirement to destroy those symmetric behaviors. Furthermore, as our forgery attack shows, evaluating the differential properties in a straightforward manner (providing the best in terms of probability differential characteristic), does not guarantee security against differential attacks in the nonce-repeating mode.

We would also like to emphasize the importance of the technique used in the forgery attack on the nonce-repeating mode. Due to the mode and the attack framework, there is no need to provide a valid tag at the beginning of the attack (forgery or state recovery). Hence the attacker can focus only on finding a differential characteristic that will leak differences in state words sufficient for recovery based on solving differential equations. The characteristic does not necessarily need to hold with a high probability, but for the forgery on PAES this was required in the first two rounds only because there was an alternative path that does not permit state recovery. In general, the probability of the characteristic is irrelevant, however, it is important for the characteristic to leak input and output differences of non-linear operations which subsequently will be used to recover the state bits. We believe that this technique (improved or modified variants) can be a valuable approach for cryptanalysis of other CAESAR submissions and authenticated encryption schemes.

Acknowledgments

The first, second, and fourth authors are supported by the Singapore National Research Foundation Fellowship 2012 NRF-NRFF2012-06.

References


Jérémy Jean received M.Sc. from the Grenoble Institute of Technology and from the Joseph Fourier University in 2010, and Ph.D. from the Ecole Normale Supérieure in Paris in 2013. His main research topics are design and analysis of symmetric-key primitives. Jérémy received the best paper award at FSE 2012.

Ivica Nikolić received M.Sc. from the Lomonosov Moscow State University, and Ph.D. from the University of Luxembourg. His main research topic is analysis of symmetric-key primitives. Ivica received the best paper award at ASIACRYPT 2010.

Yu Sasaki received the B.E., M.E. and Ph.D. from The University of Electro-Communications in 2005, 2007, and 2010. Since 2007, he has been a researcher at NTT Secure Platform Laboratories. His current research interests are in cryptography. He was awarded a paper prize from SCIS 2007 and IEICE Trans. in 2009. He also received a best paper award from IWSEC 2009, SECRIPT 2012, and IWSEC 2012.

Lei Wang received the M.E. and Ph.D. from The University of Electro-Communications in 2009, and 2011, respectively. His current research interests are in cryptography. He was awarded a paper prize from SCIS 2008 and IEICE Trans. in 2009. He also received a best paper award from IWSEC 2009 and IWSEC 2012.