IET Information Security

Research Article

IET Journals
The Institution of Engineering and Technology

DINAMITE: internal differential match-in-the-end attack on eight-round PAEQ

ISSN 1751-8709
Received on 9th February 2018
Revised 7th October 2018
Accepted on 30th November 2018
E-First on 24th April 2019
doi: 10.1049/iet-ifs.2018.5033
www.ietdl.org

Dhiman Saha[1] ✉, Sourya Kakarla[2], Dipanwita Roy Chowdhury[3]
[1]*Department of Electrical Engineering and Computer Science, IIT Bhilai, India*
[2]*Microsoft Corporation, Hyderabad, India*
[3]*Crypto Research Lab, Department of Computer Science and Engineering, IIT Kharagpur, India*
✉ *E-mail: crypto@dhimans.in*

Abstract: The authors explore a cryptanalysis strategy which seems to be particularly applicable to parallelisable ciphers where the key forms a part of the internal state. The proposed technique combines internal differentials with guess and determine analysis to come up with what is referred to as the match-in-the-end attack. The notion of difference here deviates from the classical differential where the difference is controllable via the plaintext/ciphertext. Here, they exploit the Hamming distance between parallel branches to devise the differential trail. They apply the strategy on *full* eight (out of 20) rounds of parallelisable authenticated cipher [parallelisable AE based on quadrupled AES (PAEQ)] to devise key recovery attacks with practical time complexities. They first show an initial attack on paeq-64/80/128 and then devise improvements which give us the best key-recovery attacks with time complexities of $2^{33}, 2^{48}$, and 2^{64}, respectively. While the best reported attacks on eight-round paeq-64/80/128 have a data complexity of 2^{89} blocks, the result improves their time complexities by factors of $2, 2^{18}$, and 2^{34}, while preserving the data complexity. Finally, they present a nonce-based differential attack which works on paeq-128-t with 2^{64} time complexity but uses just two single block known plaintexts making it the most practical attack on any round-reduced PAEQ variant reported so far.

1 Introduction

Authenticated encryption (AE) is a relatively new area in symmetric-key crypto which has started receiving wide-spread attention in the recent past. This is particularly attributed to the fact that since its inception [1, 2] the design of AE schemes has lacked organised effort from the crypto community. The reports of serious vulnerabilities [3, 4] in OpenSSL and TLS testify this fact highlighting the need for a methodical treatment of the problem. The problem AE tries to address is to amalgamate the two different cryptographic goals of privacy and integrity into a *unified* as well as *efficient* crypto primitive. The announcement of the competition for authenticated encryption: security, applicability, and robustness (CAESAR) [5] competition marked the first concerted effort to explore, analyse and finally standardise a portfolio of authenticated ciphers and follows the likes of successful competitions such as advanced encryption standard (AES) [6] and secure hash algorithm 3 (SHA3) [7]. The competition has seen many innovative submissions (57 accepted) with unique designs which not only satisfy the known desirable properties but also identify new ones. The competition witnessed 30 designs progressing to round 2 out of which 14 made it to the on-going third round.

In this work, we try to explore a cryptanalytic strategy which tries to combine two well-known techniques: internal differentials (IDs) and guess-and-determine (GAD). The idea of exploiting differentials within a cipher is inspired from the prospect of internal symmetry of a construction and has been popular in cryptanalysis of symmetric-key designs [8, 9]. In this regard, it is apparent that parallelisable ciphers exhibit nice internal symmetry between the states of the parallel branches. On the other hand, GAD is a simple but effective technique where, as the name suggests, the idea is to guess a part of the state to determine another part primarily exploiting low diffusion. It has found some recent applications in cryptanalysis of some CAESAR candidates [10–12]. The current work tries to simultaneously use both the above techniques to arrive at a kind of match-in-the-end (MITE) scenario. This deviates from the classical meet-in-the-middle (MITM) case where elimination generally takes place in the middle. We call this strategy DINAMITE which is shorthand for (**D**)iffer (**In**)ternally (**A**)nd (**M**)atch-(**I**)n-(**T**)he-(**E**)nd and intend to showcase this as a generic tool specially for parallelisable ciphers in the counter-mode where *the key, either directly or indirectly, forms a part of the internal state* under certain assumptions. For the case-study, we choose parallelisable AE based on quadrupled AES (PAEQ) [13], which was a round 2 candidate of CAESAR. It was introduced by Biryukov and Khovratovich [14] as a parallelisable and on-line authenticated cipher along with a generic mode of operation [parallelisable permutation-based AE (PPAE)] which calls a fixed-length public permutation. PAEQ is actually a concrete PPAE instantiation with a 512-bit wide AES-based internal permutation called AESQ. For the CAESAR submission [13], the authors proposed many variants of PAEQ which were primarily segregated into two sets: primary and secondary. While the primary sets (paeq-64/80/128) provides corresponding security levels of 64/80/128 bits, the secondary sets constitute variants with higher security levels and additional features such as quick tag update and protection against nonce-misuse. By construction, PAEQ (or more generally PPAE) meets the requirements we mentioned above and hence forms a prime candidate for demonstrating a proof-of-concept. Moreover, as PAEQ instantiates an AES-based internal permutation, it automatically magnifies the impact and scope of the results presented here. Our contributions are summarised as follows:

- Identification of a technique (DINAMITE) based on IDs and GAD approach for possibly analysing a class of parallelisable ciphers in the counter mode under appropriate settings.
- Demonstrating the attack for complete key-recovery of full eight rounds of paeq-80/128.
- Improving the attack to get practical time complexities for all the three variants paeq-64/80/128.
- Further tweaking the approach for paeq-128 and proposing a new nonce-based differential approach for paeq-128-t to achieve practical time as well as data complexities.

Table 1 Different PAEQ variants

	PAEQ	|Key|	|Nonce|	|Tag|	Security	Extra features
primary sets	paeq-64	64	64	64	64-bit	
	paeq-80	80	80	80	80-bit	
	paeq-128	128	96	128	128-bit	
secondary sets	paeq-64-t	64	64	512	64-bit	quick tag update nonce-misuse + tag update quick tag update nonce-misuse + tag update
	paeq-64-tnm	64	128	512	64-bit	
	paeq-128-t	128	128	512	128-bit	
	paeq-128-tnm	128	256	512	128-bit	
	paeq-192	192	128	128	128-bit	
	paeq-160	160	128	160	160-bit	
	paeq-256	256	128	128	128-bit	

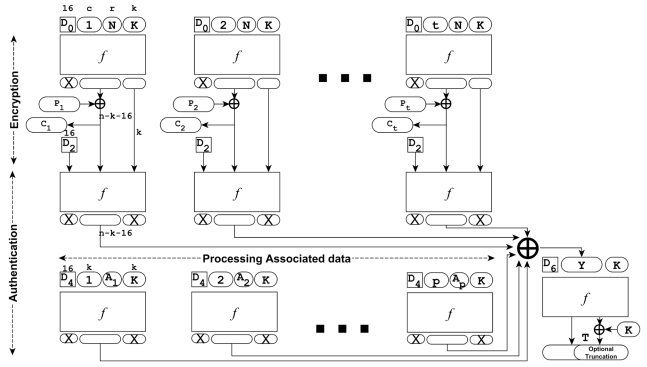

Fig. 1 *Encryption and authentication with PAEQ [11]*

- Achieving the best complexities of $2^{33}, 2^{48}$, and 2^{64} for key-recovery attacks on eight rounds of paeq-64/80/128, respectively.

1.1 Outline

The rest of the paper is organised as follows. Section 2 briefly describes PAEQ along with its mode of operation PPAE and the internal permutation AESQ. The generic idea of the proposed DINAMITE attack is presented in Section 3 while the proof-of-concept in the form of a real attack on eight-round PAEQ and further improvements are incrementally elaborated in Section 4. A discussion on comparative analysis with the best key-recovery attack known on PAEQ is furnished in Section 5 while the concluding remarks are given in Section 6.

2 PAEQ specifications

As stated earlier, it was proposed by Biryukov and Khovratovich[14] and also submitted to CAESAR where it made up to round 2 before being eliminated in round 3. It features a new generic mode of operation PPAE. PAEQ has been designed to achieve a security level equal to the key length. The authors credit its simplistic design to the permutation-based construction. It is fully parallelisable and on-line and offers a security level up to 128 bits and higher (up to $w/3$, where w is the width of internal permutation). It was mentioned before that the different members of the PAEQ family are classified into two sets based on the security level and extra features as shown in Table 1. Being parallelisable, the PPAE separates its branches based on a counter and resembles the counter mode of operation. In the subsequent subsections, we touch upon the basics of $\mathtt{PPAE}_\mathtt{f}$ and the internal permutation AESQ and refer an interested reader to [13, 14] for details.

2.1 PPAE mode of operation

PPAE is the generic mode of operation instantiated with an n-bit permutation f. It is designed to be truly parallel with encryption and authentication operating simultaneously. The inputs to the permutation for each parallel branch are formatted as follows:

$$\text{Plaintext block} \rightarrow D_i \parallel \texttt{counter} \parallel \texttt{N} \parallel \texttt{K},$$

$$\text{Associated data (AD) block} \rightarrow D_i \parallel \texttt{counter} \parallel \text{AD} - \text{block} \parallel \texttt{K},$$

where D_i is the domain separator, N is the nonce, and K is the key. The plaintext and AD are divided into blocks of size $n-k-16$ and $n-2k-16$, respectively, where k is the key-size and n is the width of the permutation. Incomplete last blocks are padded using the byte-length of the block and domain separators are changed accordingly. Plaintext processing and authentication call f twice while AD data is authenticated using a single call. Partial authentication data from all branches are passed to a final call to f, the output of which is optionally truncated to get the tag. The entire operation is depicted in Fig. 1 while the procedure is illustrated in Algorithm 5 (Fig. 2) in the Appendix. It can be noted that the key is directly absorbed in the input to each branch, which aligns well with the philosophy of the technique introduced in this work.

2.2 AESQ: the core permutation

To describe AESQ we reuse the definition of the word, state, and substate introduced by Saha and Roy Chowdhury [15].

Definition 1 (word): Let $\mathbb{T} = \mathbb{F}[x]/(x^8 + x^4 + x^3 + x + 1)$ be the field $\mathbb{F}_{2^8}$ used in AES MixColumns operation. Then a *word* is defined as an element of $\mathbb{T}$. [A word if the unknown is denoted by 'X'.]

Definition 2 (sub-state, state): The internal *state* (s) of the AESQ permutation is defined as a four-tuple of sub-states (s^1, s^2, s^3, s^4) where each sub-state ($s^m = [s^m_{i,j}]$) is a (4×4)-word matrix

$$s^m = [s^m_{i,j}], \quad \text{where} \quad \begin{cases} s_{i,j} \in \mathbb{T} \cup \{'X'\}, \\ 0 \le i, j < 4; \quad m \in \{1,2,3,4\}, \end{cases}$$
$$s = (s^1, s^2, s^3, s^4).$$

Here, m denotes the sub-state index, i.e. the relative position of the sub-state inside the state. A column of $[s^m_{i,j}]$ is denoted by $s^m_{*,j}$, while a row is referred to as $s^m_{i,*}$.

AESQ, which implies quadrupled AES, operates on four 128-bit substates in parallel applying on them a round function that is almost the same as AES. After every 2 rounds, a Shuffle is initiated, which basically permutes the columns of all the substates as per Table 2. The complete permutation is a composition of 20 rounds. We now give some notations that we will be using throughout this work.

2.2.1 Notations: The rth round of AESQ operating on internal state s is visualised as $\mathscr{R}_r(s) = \mathscr{R}^1_r(s^1) \parallel \mathscr{R}^2_r(s^2) \parallel \mathscr{R}^3_r(s^3) \parallel \mathscr{R}^4_r(s^4)$, where $\mathscr{R}^m_r(s^m)$ captures the individual AES-like rounds operating on each substate. So, $\mathscr{R}^m_r = \alpha^m_r \circ \mu^m_r \circ \rho^m_r \circ \beta^m_r$, where $\beta^m_r, \rho^m_r, \mu^m_r$ and α^m_r represent the standard AES round operations SubBytes, ShiftRows, MixColumns, and AddRoundConstants,

Input: $P \leftarrow$ Plaintext, $N \leftarrow$ Nonce, $|N| = r$, $K \leftarrow$ Key, $|K| = k$, $A \leftarrow$ Associated Data, $f \leftarrow$ Internal permutation, $n \leftarrow$ Internal state size
Output: $C, T \rightarrow$ Ciphertext and Tag

$D_i = (k, (r+i) \mod 256),\ i = 1, 2, \cdots, 6$ ▷ Generating 2-byte domain separators
$\{P_1, P_2, \cdots, P_t\} \leftarrow P \quad |P_i| = (n-k-16)$ bits
$\{A_1, A_2, \cdots, A_p\} \leftarrow A \quad |A_i| = (n-2k-16)$ bits
if $(|P_t| < n-k-16)$ **then**
  $P_t \leftarrow P_t||a||a\cdots||a$ ▷ $a = |P_t|/8$ and $|a| = 1$ byte
if $(|A_p| < n-2k-16)$ **then**
  $A_p \leftarrow A_p||b||b\cdots||b$ ▷ $b = |A_p|/8$ and $|b| = 1$ byte
$Y = \mathbf{0}$ ▷ $|Y| = n-k-16$
for $i = 1$ to t **do**
  $V_i \leftarrow D_0||R_i||N||K$
  ▷ $R_i \leftarrow$ Branch Index, $R_i = i, |R_i| = n-k-r-16$; $D_0 \leftarrow D_1$ for incomplete last block
  $W_i \leftarrow f(V_i)$;
  $C_i \leftarrow W_i[17\cdots(n-k)] \oplus P_i$
  $X_i \leftarrow D_2||C_i||W_i[(n-k+1)\cdots n]$ ▷ $D_2 \leftarrow D_3$ for incomplete last block
  $Y_i \leftarrow (f(X_i))[17\cdots(n-k)]$;
  $Y \leftarrow Y \bigoplus Y_i$
for $i = 1$ to p **do** ▷ Binding Associated Data
  $X'_i \leftarrow D_4||R_i||A_i||K$
  ▷ $R_i = i,\ |R_i| = k$; $D_4 \leftarrow D_5$ for incomplete last block
  $Y'_i \leftarrow (f(X'_i))[17\cdots(n-k)]$
  $Y \leftarrow Y \bigoplus Y'_i$
$T \leftarrow f(D_6||Y||K) \oplus (0^{n-k}||K)$
$C = \{C_1, C_2, \cdots, C_t\}$ ▷ Truncate C_t for incomplete last plaintext block

Fig. 2 *Algorithm 5:* $\mathtt{PPAE_f(P,N,K,A,n)}$ *[11]*

Input: $s = (s^1, s^2, s^3, s^4) \leftarrow$ Input State
Output: Updated $s \leftarrow$ Output State

for $r = 1 : 20$ **do**
  for $m = 1 : 4$ **do**
    $rc_r^m = ((r-1) * 4 + m)$
    $s^m \leftarrow \alpha_r^m \circ \mu_r^m \circ \rho_r^m \circ \beta_r^m(s^m)$ ▷ $s^m_{1,*} \xleftarrow{\alpha_r^m(s^m)} s^m_{1,*} \oplus \{rc_r^m, rc_r^m, rc_r^m, rc_r^m\}$
  if $r \mod 2 = 0$ **then**
    $s \leftarrow \mathscr{S}(s)$
return s

Fig. 3 *Algorithm 1:* $\mathtt{AESQ(s)}$

respectively. While referring to these operations with respect to the entire state we drop the substate-index and denote them as β_r, ρ_r, μ_r, and α_r. The shuffle at the end of every two rounds is represented by $\mathscr{S}$. The round constant for sub-state m in round r of AESQ is given by $rc_r^m = ((r-1)*4+m)$. In α_r^m, rc_r^m is added to all words of row $s^m_{1,*}$. The pseudo-code for AESQ permutation is furnished in Algorithm 1 (see Fig. 3).

We will use fractional values to refer to incomplete rounds by assuming that each round operation progresses a round by one-fourth with the exception of $\mathscr{S}$, which we will choose to ignore for this calculation. So, by 5.25 rounds we mean that we have reached up to β_6. Finally, in this work, we will be dealing with eight-round PAEQ, which instantiates eight-round AESQ referred to as $\mathtt{AESQ}^8 = \mathscr{S} \circ \mathscr{R}_8 \circ \mathscr{R}_7 \circ \cdots \circ \mathscr{S} \circ \mathscr{R}_2 \circ \mathscr{R}_1$.

Table 2 Column mapping under shuffle ($\mathscr{S}$)

From	$s^1_{*,0}$	$s^1_{*,1}$	$s^1_{*,2}$	$s^1_{*,3}$	$s^2_{*,0}$	$s^2_{*,1}$	$s^2_{*,2}$	$s^2_{*,3}$	$s^3_{*,0}$	$s^3_{*,1}$	$s^3_{*,2}$	$s^3_{*,3}$	$s^4_{*,0}$	$s^4_{*,1}$	$s^4_{*,2}$	$s^4_{*,3}$
To	$s^1_{*,3}$	$s^4_{*,3}$	$s^3_{*,2}$	$s^2_{*,2}$	$s^1_{*,1}$	$s^4_{*,1}$	$s^3_{*,0}$	$s^2_{*,0}$	$s^1_{*,2}$	$s^4_{*,2}$	$s^3_{*,3}$	$s^2_{*,3}$	$s^1_{*,0}$	$s^4_{*,0}$	$s^3_{*,1}$	$s^2_{*,1}$

2.3 Previous cryptanalysis of PAEQ

The designers of PAEQ have furnished a comprehensive cryptanalysis in [13]. They mount an eight-round constrained input constrained output (CICO) attack with a complexity of 2^{32} assuming that the last 384 bits of the input and the output of AESQ are known. A rebound-attack is also presented to devise a 12-round distinguisher on AESQ with a complexity of 2^{256}. The first third-party cryptanalysis was published by Bagheri *et al.* [16]. They reported improved rebound attacks on 12-round AESQ reducing the time complexity to 2^{128} with negligible memory. They further extend the rounds to 16 with a computational overhead of 2^{192} and 2^{128} memory. Additionally, time-memory trade-offs and multiple limited birthday distinguishers are also discussed. In [15], Saha and Roy Chowdhury reported an ID fault attack on primary sets of PAEQ using two faults while retrieving the key with complexities of $2^{16}, 2^{16}, 2^{50}$ for full-round `paeq-64/80/128`, respectively. A GAD key-recovery attack was reported in [11] on six, seven, eight out of the 20 rounds of PAEQ with practical complexities. However, the eight-round attack dropped the last $\mathscr{S}$ as the authors argued that to be a requirement for their approach. This limitation was overcome in [12] by the same authors using an internal–differential approach at the cost of higher data complexity. The state-of-the-art of PAEQ cryptanalysis is summarised in Table 3.

3 DINAMITE: differ internally and MITE

Our primary focus is to look at parallelisable ciphers in the counter mode where key forms a part of the internal state which is fed to some permutation. Since, in such ciphers, any intermediate state recovery might trivially imply key-recovery, so the output is generally truncated by an amount greater than or equal to the key-size. If we look at CAESAR submissions, examples of such authenticated ciphers include π-cipher [18], NORX [19, 20] and PAEQ [13, 14]. One common aspect is that once the key is absorbed into the state (either directly like in PAEQ or indirectly through π-function in π-cipher and init function in NORX), there is no interaction with the key before the cipher-text blocks are released. [The key might again interact with a state like in PAEQ and NORX but it happens just before the tag generation.] It is thus apparent that a state-recovery would lead to a key-recovery. It can be noted that the parallel branches (lanes in case of NORX) are separated only by a counter signifying low and *controllable* Hamming distance between them. In this work, we propose to exploit the counter differences to devise ID trails. Furthermore, we want to target the truncation by employing GAD analysis. This has been used by Boura *et al.* in case of π-cipher [10], where they exploit the low diffusion of the π-function. Our idea is to intelligently guess the part/whole of the truncated output of one of the branches and use the differential trail and the known part of the output of the other branch to verify this guess. This is a kind of MITE scenario as the elimination takes place at the end. The idea is depicted in a generic sense in Fig. 4. We call this strategy DINAMITE which is shorthand for **D**iffer **In**ternally and MITE. In the rest of the work, we demonstrate the application of this strategy on PAEQ to mount key-recovery attacks.

4 DINAMITE attack on eight-round PAEQ

In this section, we present an instance of the technique on PAEQ. It can be recalled that the output of the first call to AESQ during

Table 3 Present cryptanalytic results on PAEQ

Attack type	#$\mathscr{R}$	Complexity				Ref.
		Time			Memory	
		AESQ				
CICO	8	2^{32}				[13]
rebound attack	12	2^{256}			2^{256}	[16]
	12	2^{128}			Negligible	
	16	2^{192}			2^{128}	
time-memory trade-off	12	$2^{102.4}$			$2^{102.4}$	
Multi Ltd.- birthday distinguisher	16	2^{188}			2^{128}	
		paeq-64	paeq-80	paeq-128	Data (# KP)	
fault attack	Full	2^{16}	2^{16}	2^{50}	1 (255 B)	[15]
	Full	2^{16}	2^{16}	2^{50}	1 (2^{8k} B)	[17]
guess and determine	6	1	2^{16}	2^{32}	1 (1 B)	[11]
	7	2^{24}	2^{32}	2^{40}		
	8[a]	2^{48}	2^{48}	2^{48}		
ID	8	2^{34}	2^{66}	2^{98}	1 (2^{89} B)	[12]

[a]Drops last shuffle; KP, known plaintexts; B, blocks; k, depends on the fault model.

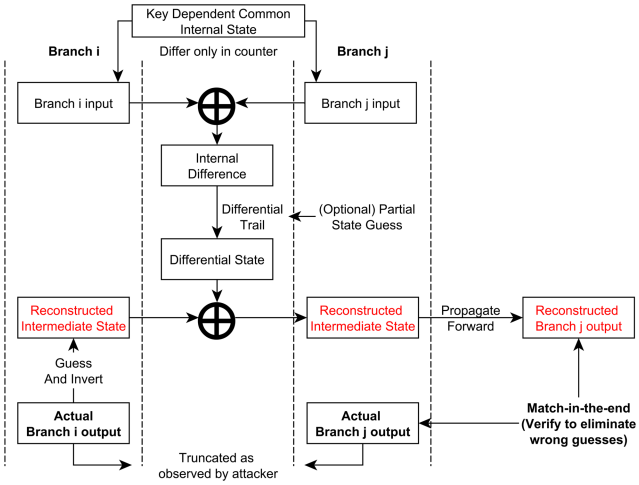

Fig. 4 *Demonstration of DINAMITE strategy for state-recovery in parallelisable ciphers in counter-mode. Here the key is assumed to be absorbed as part of the internal state and branch output is truncated*

encryption is truncated before being xored with the plaintext to generate the ciphertext for any parallel branch. Due to the truncation, an attacker will only observe a *part* of the internal state using the plaintext–ciphertext pair. This implies that we have to deal with states or sub-states that may have multiple unknown values. To capture this formally we use the notion of byte-entropy introduced by Saha *et al.* [11].

Definition 3: (byte-entropy [11]*):* The byte-entropy of a state/sub-state, denoted by $\mathscr{E}$, is defined as the number of unknown bytes in the state/sub-state

$$\mathscr{E}(s^m) = \left|\{s^m_{i,j} =' X', \forall i, j\}\right|,$$
$$\mathscr{E}(s) = \sum_{\forall s^m \in s} \mathscr{E}(s^m).$$

Consequently, the byte-entropy of a completely specified state/sub-state is zero. It was argued in [11] why the only operation that affects the byte-entropy of a sub-state and thereby a state is MixColumns. We now present an observation on PAEQ that will be vital in formulating the DINAMITE attack that is proposed later.

4.1 Observations on PAEQ

The following observation tracks the diffusion of bytes belonging to *different* substates while inverting multiple rounds of AESQ. This actually exploits the MegaSBox property (see Fig. 5) discussed by the designers of PAEQ in the submission document [13].

Observation 1: (substate independence): An attacker can independently guess all the substates at the end of $\mathscr{R}_n$ to determine corresponding parts of the entire state before α_{n-3} (or after μ_{n-3}). [It is understood that here $2|n$.]

$$s \xrightarrow{(\alpha_{n-3})^{-1} \circ (\mathscr{R}_{n-2})^{-1} \circ \mathscr{S}^{-1} \circ (\mathscr{R}_{n-1})^{-1} \circ (\mathscr{R}_n)^{-1}} r$$
$$\left\{ \begin{matrix} s^k_{(m-1,0)}, s^k_{(m,1)}, \\ s^k_{(m+1,2)}, s^k_{(m+2,3)}, \end{matrix} \forall k \right\} \text{by } \mathtt{r}^{\mathtt{m}^{\mathtt{BytesDetermined}}} r^m,$$

where r is the state after $\mathscr{R}_n$ and the subscript operations are modulo 4.

Remark 1: In order to understand this we need to focus on inputs of all the InverseMixcolumns that happen in the 3.25 rounds between $\mathscr{R}_n$ and α_{n-3}. Note that we can ignore the InverseShiftRows and $\mathscr{S}^{-1}$ since they just permute the bytes and columns, respectively. The following discussion uses the case for $n = 8$. See Fig. 6 for a pictorial illustration where bytes at the end of $\mathscr{R}_8$ are uniquely numbered and coloured based on which substate they belong to.

- $(\mu_8)^{-1}$ *and* $(\mu_7)^{-1}$*:* The input columns originate from the same substate. So diffusion is limited to each substate.
- $(\mu_6)^{-1}$*:* Due to $(\mathcal{S})^{-1}$, columns among substates have been mixed. However, the column-wise inputs still originate from individual substates. So, the diffusion at the end of $(\mu_6)^{-1}$ is localised to the bytes originating from the substates at the end of $\mathscr{R}_8$.
- $(\mu_5)^{-1}$*:* This is the first `InverseMixcolumns` where input columns are heterogeneous in nature, i.e. individual bytes belong to different substates. Hence, in the backward direction, the first actual inter-substate diffusion is credited to $(\mu_5)^{-1}$.

In Fig. 6, we can easily observe that the bytes at the input of α_5 can be segregated, each of them depending only on the value of the corresponding substate after $\mathscr{R}_8$ that we started with. Thus, in the general case, the above observation holds up to the input of α_{n-3} with $(\mu_{n-3})^{-1}$ destroying the substate independence.

An interesting implication of this observation is that we can determine the bytes of the state before α_5 by independently guessing the corresponding (unknown) bytes of substates after $\mathscr{R}_8$. Subsequently, the complexity of this GAD step is bounded by the complexity of guessing the substate with the maximum byte entropy.

Proposition 1: The GAD complexity for the state before α_{n-3} based on the state (say r) after $\mathscr{R}_n$ is dominated by the substate(s) at the end of $\mathscr{R}_n$ which have the maximum byte-entropy and is given by

$$2^{8\times v} \quad \text{where} \quad v = \max_{\forall r^m \in r} \mathscr{E}(r^m).$$

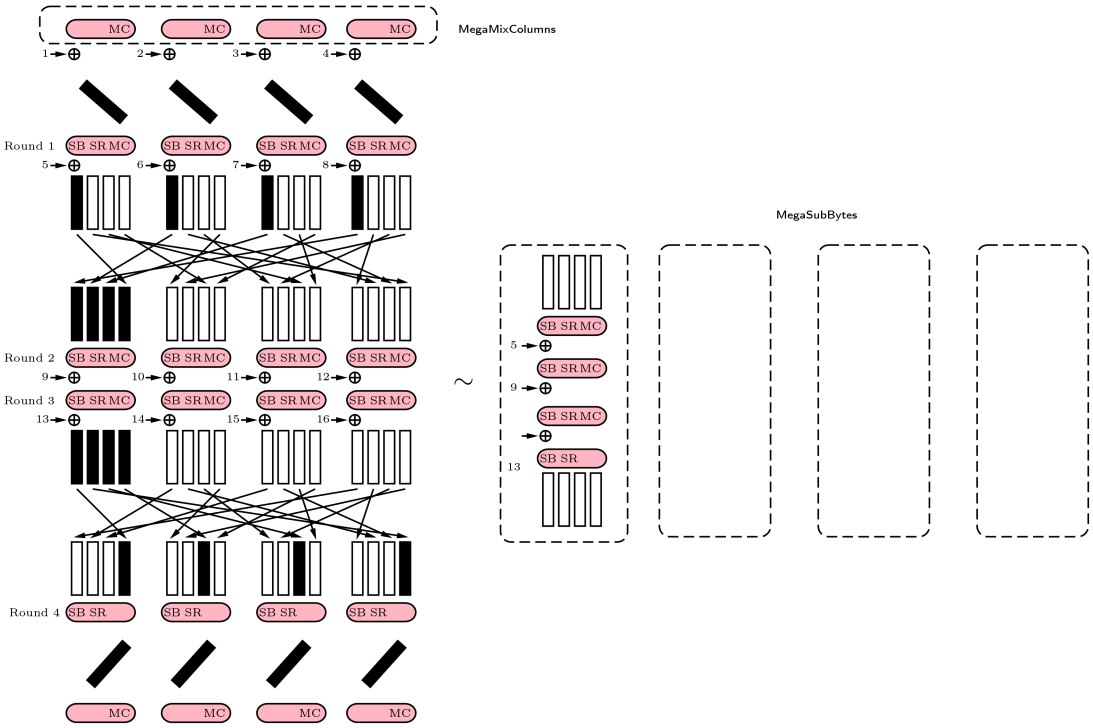

Fig. 5 *MegaSBox in* `AESQ` *[13]*

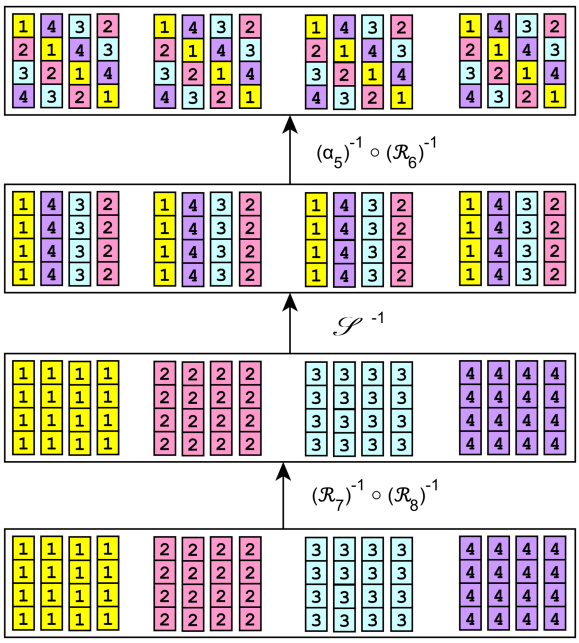

Fig. 6 *Demonstration of 'no' inter-substate diffusion for 3.25 rounds. Bytes marked* $x(1 \le x \le 4)$ *after* $\mathscr{R}_8$ *uniquely determine bytes marked x before* α_5

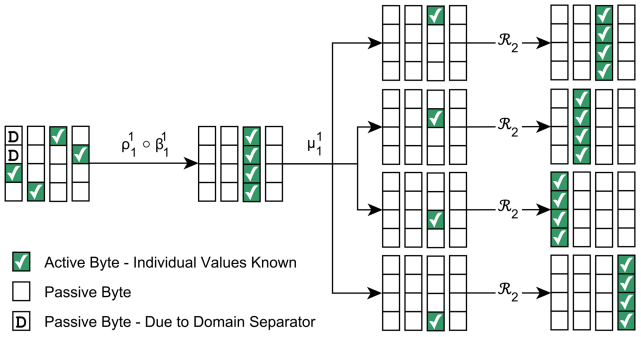

Fig. 7 *Possible transitions for differences in counter values in the first substate [12]*

Table 4 Conforming counter values for transitions like in Fig. 7 [12]

(a) Counter 1 values			
40	00	00	00
40	00	00	00
01	00	00	00
00	00	00	00

(b) Possible counter 2 values							
40	00	3a	00	40	00	74	00
40	00	00	01	40	00	00	01
7e	00	00	00	e0	00	00	00
00	28	00	00	00	23	00	00
40	00	09	00	40	00	92	00
40	00	00	01	40	00	00	01
18	00	00	00	88	00	00	00
00	da	00	00	00	1a	00	00

It must be noticed that we are not claiming to retrieve the actual state before α_5 but rather to determine independent parts of the state based on their one-to-one correspondence with the substates after $\mathscr{R}_8$.

4.2 Description of the basic attack

It was earlier stated that Saha *et al.* reported key recovery attacks on *pseudo* [where the last shuffle was dropped] eight rounds [11] and extended it to full eight rounds using an ID strategy [12]. The attack proposed by them progressed in three phases: forward, backward, and MITM. The main idea was to recover the fourth substate of the state after the second round which would lead to a key recovery. Their attack recovers the substate column-wise repeating a basic strategy four times. In contrast, our idea is to *recover the entire internal state* which in turn recovers the key. Our attack, which we refer to as DINAMITE proceeds in three phases namely: ID phase, GAD phase, and the MITE phase.

4.2.1 ID phase: This phase resembles the forward phase of the GAIN$_8^\star$ attack in [12]. However, we will need only one trail in comparison with four trails required in [12]. Despite the resemblance, we restate this phase below for brevity.

Our aim here is to introduce an internal difference and for this we will rely on the counter bytes. The PPAE mode of operation of PAEQ dictates that any two branch-inputs differ only in the counter bytes. We will propagate the differences in the counter bytes to devise a differential trail.

- The first task is to choose counter values in such a way that the difference between them gives $4 \to 1 \to 4$ transition after the first two rounds for the first substate. There are four possible transitions of this type while keeping the counter values minimum [note that no difference can be induced in the first two bytes which represent the domain separator bytes], which are depicted in Fig. 7.
- For the current attack, we need to induce *any one* of the four possible transitions. It was shown by Saha *et al.* [12] that the $4 \to 1 \to 4$ trail is attained for free by carefully manipulating the counter values. This is demonstrated by Table 4 which gives candidate counter values that lead to the required transitions without incurring any overhead.
- To grasp the rest of the trail, we need to look at the active substate after the first application of $\mathscr{S}$. Recall that since there was only one active column after $\mathscr{R}_2$, we have exactly one active substate after $\mathscr{S}$. To make precise statements, we refer to Fig. 8. The active substate has the property that it has one unknown passive column, one known active column, and two known passive columns. By *known,* we mean that the individual values that lead to the activeness (passiveness) are known. Also, the unknown passive column originates from the fourth substate after $\mathscr{R}_2$.
- Guessing the bytes of the unknown passive column helps to propagate $\mathscr{R}_3$ and $\mathscr{R}_4$ followed by $\mathscr{S}$. Now, since the original contributing values of the active bytes are known, we can also penetrate β_5 thereby reaching the input of μ_5. The linearity of MixColumns allows us to apply μ_5 on the differential and get the output. Though the individual values are no longer known, the difference is known for the entire state.

This phase gives us a five-round trail with the output being the ID state after μ_5. Algorithm 2 (see Fig. 9) gives a procedural illustration of this phase of the attack. In the next phase, we try to reduce the byte-entropy of the output.

4.2.2 GAD phase: This phase tries to capitalise on Observation 1. Let the branches determined by the counter values used in the previous phase be i and j. Then the attacker knows $P_i \oplus C_i$ and $P_j \oplus C_j$. He uses the partially known state $\mathscr{S}^{-1}(P_i \oplus C_i)$. Recall that since this is an eight-round attack $\mathscr{S}^{-1}(P_i \oplus C_i)$ corresponds to the

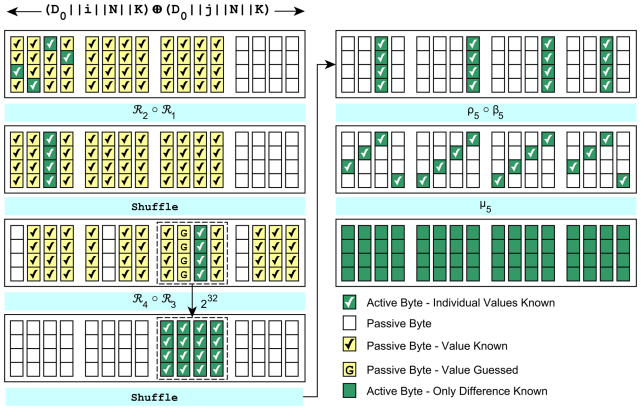

Fig. 8 *4.75-round ID trail using differences in counter bytes*

Input: $\begin{cases} i, j \rightarrow \text{Branch indices (Counter-Values)} \\ g \rightarrow \text{Column Guess} \end{cases}$
Output: $\{$Internal Differential State after μ_5

1: $v \xleftarrow{\text{AESQ}^2} D_0||i||N||\mathbf{X}$
2: $w \xleftarrow{\text{AESQ}^2} D_0||j||N||\mathbf{X}$
3: $e \leftarrow v \oplus w$
▷ e has exactly one active substate
4: $e^m \xleftarrow{\texttt{Get Active Substate}} e$
▷ e^m has exactly one unknown passive column
5: $e^m_{c,*} \xleftarrow{\texttt{Get Passive Column}} e^m$
▷ $e^m_{c,*} = v^m_{c,*} \oplus w^m_{c,*} = \mathbf{0}$
6: $v^m_{c,*}(w^m_{c,*}) \xleftarrow{\texttt{Assign Guessed Value}} g$
7: $v, w \xleftarrow[\texttt{Using Guessed Column}]{\rho_5 \circ \beta_5 \mathscr{S} \circ \mathcal{R}_4 \circ \mathcal{R}_3} v, w$
8: $f \xleftarrow[\texttt{Get Internal Differential State}]{\mu_5} (v \oplus w)$
9: **return** f

Fig. 9 *Algorithm 2: GETINTDIFFSTATE* (i, j, g)

Input: $\begin{cases} s \rightarrow \text{Output of } \mathcal{R}_8 \text{ for branch } i \\ f \rightarrow \text{Internal Differential State} \end{cases}$
Output: Reconstructed $\mathcal{R}_8$ output for branch j

1: **Independently** Guess Unknown Bytes of each $s^m \in s$
2: $s \xleftarrow[\texttt{Using Guessed Values}]{(\alpha_5)^{-1} \circ (\mathcal{R}_6)^{-1} \circ \mathscr{S}^{-1} \circ (\mathcal{R}_7)^{-1} \circ (\mathcal{R}_8)^{-1}} s$
▷ Observation 1
3: $t \leftarrow s \oplus f$
▷ Using internal differential
4: $t \xleftarrow[\texttt{Propagating Forward}]{\mathcal{R}_8 \circ \mathcal{R}_7 \circ \mathscr{S} \circ \mathcal{R}_6 \circ \alpha_5} t$
5: **return** t

Fig. 10 *Algorithm 3:* GAD(s, f)

output of $\mathscr{R}_8$. Applying Observation 1, the state after μ_5 corresponding to branch i is determined. Now, due to the previous phase, we already have the differential state after μ_5. *So we compute the XOR of these states to reconstruct the state after μ_5 for branch j.* The next step is to propagate this state forward up to the output of $\mathscr{R}_8$. At the end of this phase, we have a candidate for each substate of $\mathscr{S}^{-1}(P_j \oplus C_j)$ based on the corresponding *independent* guess of substates of $\mathscr{S}^{-1}(P_i \oplus C_i)$. Algorithm 3 (see Fig. 10) summarises this phase.

4.2.3 MITE phase: This is the elimination step. By guessing the column in the ID phase and the substates in the GAD phase, the attacker has reconstructed the *full* state after $\mathscr{R}_8$ for branch j. He now uses his knowledge of the actual observable value of $\mathscr{S}^{-1}(P_j \oplus C_j)$ to match the known bytes thereby eliminating the wrong guesses. However, this should be noted that this is a substate by substate matching. Since the guesses for the substates in the previous phase were independent, the verification has to be independent. This verification eliminates all wrong guesses for both the phases and recovers each substate from $\mathscr{S}^{-1}(P_i \oplus C_i)$ and corresponding ones from $\mathscr{S}^{-1}(P_j \oplus C_j)$ individually. Since now the entire state after $\mathscr{R}_8$ is available we can just invert its eight rounds and retrieve the key from the 4th substate of the input. Fig. 11 shows an illustration of the entire attack for `paeq-80` while Algorithm 4 (see Fig. 12) furnishes the pseudo-code for DINAMITE. Since only the known bytes of a substate are matched, the probability that a wrong guess in the ID and/or the GAD phase passes is given below

$$2^{-8\times(16-\mathscr{E}(t^m))}, \quad \text{where} \quad t \leftarrow \mathscr{S}^{-1}(P_j \oplus C_j)\,.$$

4.2.4 Complexity: The time complexity of the attack can be determined by the total amount of guessing one needs to do. For the ID part, a single column guess is required and will be the same across all the three variants of `PAEQ` and amounts to 2^{32}. For the GAD phase, by virtue of the parallel guessing, the complexity is dominated by the least known substate, i.e. the substate that has the highest byte-entropy. So the total time complexity is given by

$$2^{32+8\times v}, \quad \text{where} \quad \begin{cases} s \leftarrow \mathscr{S}^{-1}(P_i \oplus C_i), \\ v = \max_{\forall s^m \in s} \mathscr{E}(s^m)\,. \end{cases}$$

Now, the value of v for `paeq-64/80/128` is 4, 4, and 6, respectively. So the complexities stand at 2^{64} for `paeq-64`, 2^{64} for `paeq-80` and 2^{80} for `paeq-128`. Naturally, the current strategy does not amount to an attack for the 64-bit version of `PAEQ`. However, it does improve the existing results on the other variants by considerable factors. The data complexity of DINAMITE is the same as GAIN$_8\star$ and is dictated by the counter-values needed to conform to the input differential. It was argued by Saha *et al.* [12] that the message needs to contain at least 2^{89} complete blocks. This is evident from the position of the last active bit of the counter observed in Table 4.

In the next subsection, we show how we can tweak the GAD phase of the above attack to get an attack on `paeq-64` as well as highly improve the complexities for `paeq-80/128`.

4.3 Improved attack: splitting the GAD phase

The improvement involves only the backward guessing phase with the ID phase remaining unaltered. The trick is to split the GAD phase into two parts. Thus, instead of guessing *all* the substates of $\mathscr{S}^{-1}(P_i \oplus C_i)$, one would guess the ones with the minimum byte-entropy first and recover them in the MITE phase. The recovery of the remaining substates is deferred. Interestingly, we have still now overlooked the fact that in addition to substate recovery, we also recover the differential state after μ_5. Thus the ID trail is available for free. We capitalise on this while mounting the second part of the GAD phase in order to retrieve the other substates. Thus while recovering the remaining substates, the complexity is only governed by the backward guessing. So the final expression for complexity is given by as follows:

$$2^{32+8\times u} + 2^{8\times v}, \quad \text{where} \quad \begin{cases} t \leftarrow \mathscr{S}^{-1}(P_i \oplus C_i), \\ u = \min\limits_{\forall t^m \in t} \mathscr{E}(t^m), \\ v = \max\limits_{\forall t^m \in t} \mathscr{E}(t^m)\,. \end{cases}$$

Consequently, plugging in the values of u, v for `paeq-64`, the time complexity is $2^{32+8\times 0} + 2^{8\times 4} = 2^{33}$ giving the best attack available in the literature (refer to Appendix 2 for the experimental results

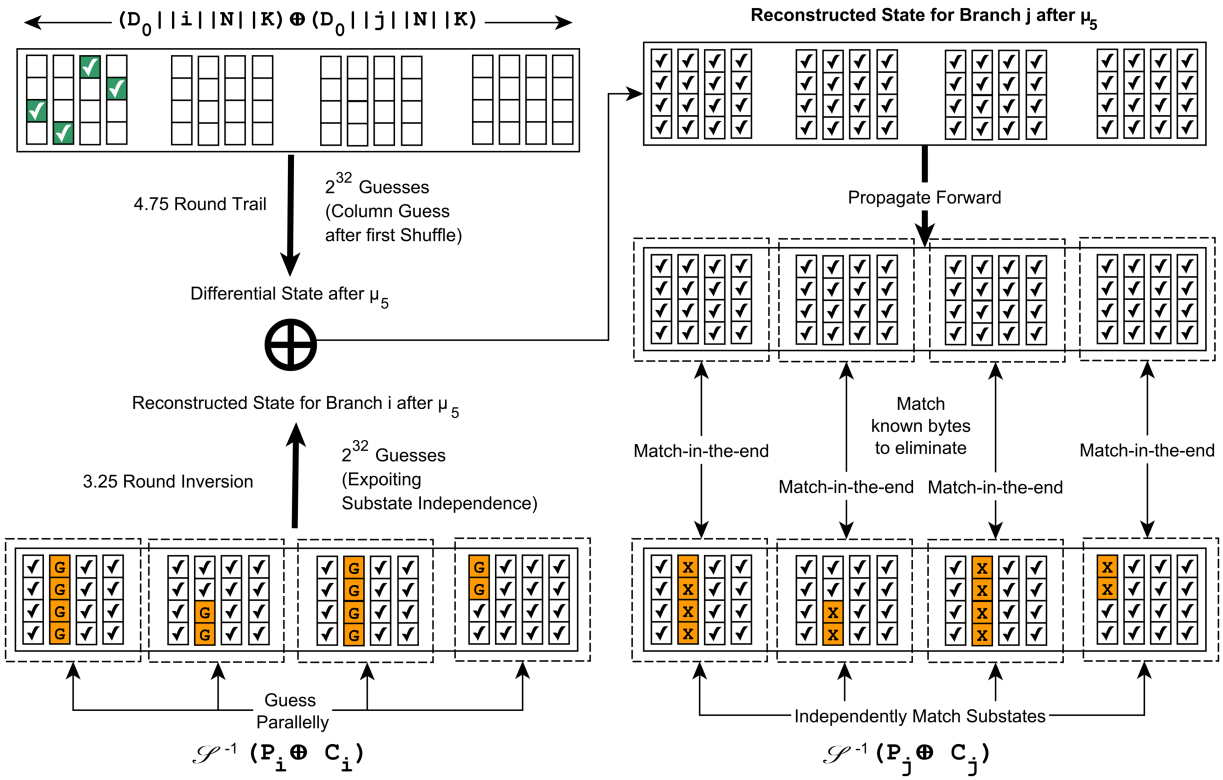

Fig. 11 *DINAMITE for eight-round* `PAEQ-80`

Input: $P, C \rightarrow$ A plaintext-ciphertext pair with at least 2^{89} complete blocks

Output: The Master Key

1: Choose any column of the 4^{th} substate after $\mathcal{R}_2$
2: Find conforming branch indices $\rightarrow i, j$ ▷ Ref. Fig. 5
3: **for** Each column guess g **do**
4: $f \leftarrow \text{GETINTDIFFSTATE}(i, j, g)$
5: $s \leftarrow \mathscr{S}^{-1}(P_i \oplus C_i)$ $t \leftarrow \mathscr{S}^{-1}(P_j \oplus C_j)$
6: $v = \max_{\forall s^m \in t} \mathscr{E}(s^m)$
7: **for** $i = 1 : 2^{8\times v}$ **do** $r \leftarrow \text{GAD}(s, f)$
8: **for** $m = 1 : 4$ **do** ▷ Match-in-the-end Step
9: **if** $t^m = r^m$ **then** ▷ Compare known values of t^m Substates are verified separately
10: Save recovered substate r^m ▷ Stop guessing s^m in Step 7
11: $r \xleftarrow[P_i \oplus C_i]{\text{Recovered State}} \mathscr{S}(r^1, r^2, r^3, r^4)$
12: $r \xleftarrow{(\text{AESQ}^8)^{-1}} r$ ▷ Get input of branch i
13: Key $\leftarrow r^4$ ▷ Retrieve Key from 4^{th} subtstate
14: **return** Key

Fig. 12 *Algorithm 4:* Dinamite(P, C)

demonstrating this attack on `paeq-64`). Accordingly, for `paeq-80`, it stands at $2^{32+8\times2} + 2^{8\times4} = 2^{48} + 2^{32}$, which can be treated as 2^{48} for all *practical* purposes while for `paeq-128` the figure is $2^{64} + 2^{48}$. However, we cannot ignore the second term in the complexity of `paeq-128`. The next subsection is dedicated to getting rid of this term and reducing the complexity to 2^{64}.

4.4 Improving further for `paeq-128`: MITM after MITE

It has to be noted that the following trick works only for `paeq-128`. The main idea is to avoid the second part of the GAD phase described in the last subsection. To understand this we need

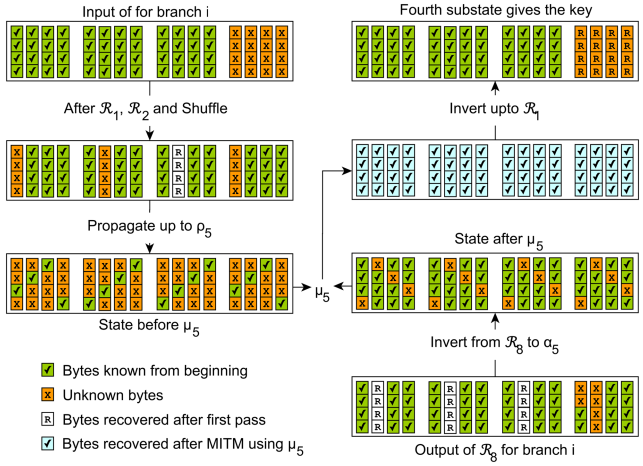

Fig. 13 *Illustration of the second pass in DINAMITE attack on* `paeq-128`

to find how may substates are recovered at the end of first GAD phase for `paeq-128`. Recall that we recover only minimum byte-entropy substates. The number of substates recovered is given by the cardinality of the set $\{t^m: \min_{\forall t^m \in t} \mathscr{E}(t^m)\}$, which is equal to three for `paeq-128`. So, using the MITE phase, three out of the four substates become completely known. Instead of trying to GAD the fourth substate, we propagate the three known substates backward up to the output of μ_5. The resultant partial state has three bytes known for each column. We now leverage on the fact that the column that was being guessed in the ID phases was also recovered while recovering the three substates. With this knowledge, we uniquely determine the partial state at the input of μ_5 with one byte known for each column (see Fig. 8). So we have a MITM scenario with the states converging on μ_5. Solving linear equations involving mixcolumns we can uniquely recover the state thereby recovering the key (see Fig. 13). The complexity is thus the same as incurred up to the first part of the GAD phase in the previous attack, which is 2^{64}.

In all the attacks reported here, the data complexity is 2^{89}, which is due to the counter values that need to be satisfied for the ID phase. Thus, despite the time complexities being practically

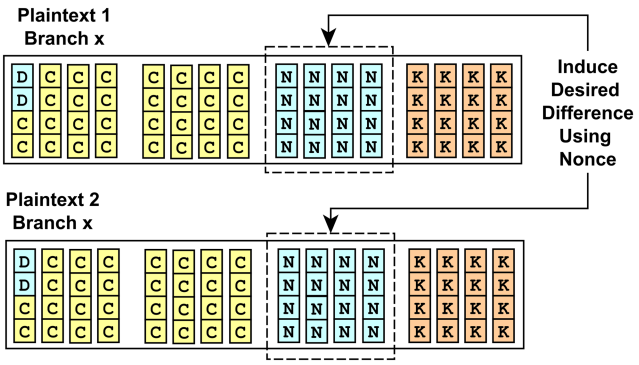

Fig. 14 *Input configuration for attack on* `paeq-128-t`

Table 5 DINAMITE attack overview

Attack	Type	Strategy ($s \rightarrow$ input state to GAD)	PAEQ variant	Complexity	
				Time	Data
Section 4.2	single-pass state recovery	ID + GAD $\forall s^m$ + MITE	paeq-64	not applicable	one 2^{89} blocks plaintext
			paeq-80	2^{64}	
			paeq-128	2^{80}	
Section 4.3	two-pass state recovery	pass-1: ID + GAD s^m: min $\mathscr{E}(s^m)$ + MITE; pass-2: GAD remaining s^m + MITE	paeq-64	$2^{32} + 2^{32}$	
			paeq-80	$2^{48} + 2^{32}$	
			paeq-128	$2^{64} + 2^{48}$	
Section 4.4	partial state recovery	pass-1: same as above; pass-2: MITM	paeq-128	2^{64}	
Section 4.5	partial state recovery	pass-1: classical differential using nonce + same as above; pass-2: MITM	paeq-128-t	2^{64}	two single block plaintexts

ID, internal differential; GAD, guess-and-determine.

Table 6 Comparative study

Attack	Complexity			Remark	Ref.
	paeq-64	paeq-80	paeq-128		
GAIN8	2^{48}	2^{48}	2^{48}	drops last $\mathscr{S}$	[11]
GAIN$_8\star$	2^{34}	2^{66}	2^{98}	full eight-round attack	[12]
DINAMITE	2^{33}	$\simeq 2^{48}$	2^{64}		Section 4

feasible, data complexity is quite high. In the next subsection, we present a technique to avoid this. However, it is applicable only on PAEQ variants where the nonce-size is >128 bits and it at least occupies an entire substate. The only PAEQ variant fitting this requirement is `paeq-128-t`.

4.5 Practical nonce-based differential attack on `paeq-128-t`

The only difference between this attack and the one described in the last subsection is the way the differential is induced. In this case, instead of an ID using counter-bytes, we will devise a classical differential using the nonce bytes. For `paeq-128-t`, the nonce covers the third substate of AESQ input and the required input difference can be introduced by manipulating the nonce bytes. The input configuration is depicted in Fig. 14. The strategy to create the trail remains the same as before and so does the remaining execution with the only difference that we will be dealing with two *different* plaintexts instead of one multi-block plaintext. Consequently, time complexity is the same as before. However, since *only two single complete-block plaintexts* will suffice to mount the attack for the two calls to PAEQ, this forms the *most practical* eight-round attack on any PAEQ variant.

5 Discussion

It is important to look at DINAMITE in the light of the existing best key-recovery attacks on PAEQ. The idea of using ID trail was first proposed by Saha *et al.* in the GAIN$_8\star$ attack [12]. However, the authors resorted to a column by column recovery of the fourth substate after $\mathscr{R}_2$ which leads to the key-recovery. It must be noted that each column recovery needs a different differential trail. Moreover, in the backward trail, they guess unknown bytes of *both* the branches which compound the complexity contributing a factor of 2 in the exponent in $2^{2\times 8\times u}$, u being the minimum byte-entropy. In contrast, in the basic DINAMITE attack, using Observation 1, we concentrate on state-recovery and since the MITE strategy allows us to guess *only one* branch, we save the factor of four making the backward guessing overhead $2^{8\times v}$, v being the maximum byte-entropy. Furthermore, since we recover the entire state in one pass we require only one ID trail and save an additional factor of four in comparison with GAIN$_8\star$ which repeats the column recovery four times. Thus, though $v > u$, we register an improvement by a factor of $2^{2+8\times(2\times u - v)}$, which translates into better attacks for `paeq-80/128`. Finally, in the modified DINAMITE attack, we are able to change v to u bettering the factor to $2^{2+8\times(2\times u-u)} = 2^{2+8\times u}$ with additional work of $2^{8\times v}$. The last enhancement gets rid of $2^{8\times v}$ for `paeq-128`. So the final factors of improvement over GAIN$_8\star$ stand at 2, $\simeq 2^8$, and 2^{32} for `paeq-64/80/128`, respectively. For all the improvements reported, DINAMITE incurs no additional overhead on the data complexity. As a matter of fact, we also introduce a scenario where the nonce is exploited to mount the same attack on `paeq-128-t` while drastically reducing the data complexity to two single complete block plaintexts. Table 5 summarises the results reported in the current work, while Table 6 furnishes the comparative figures for the key-recovery attacks reported on eight-round PAEQ.

It is interesting to note that the attack described here is independent of how the parallelism in PAEQ is actually implemented in practice. In fact, the results reported would hold

even if PAEQ is implemented in a serial manner. For the ID attack, the only constraint is a *single* message of at least 2^{89} blocks while for the nonce-based differential attacks we need *two* single block messages. The attacks basically exploit the mode of operation, which for PAEQ, is analogous to the counter mode. The Hamming distance between the inputs to the permutation is only due to the counter. Hence, specific branches are chosen, so as to induce an intended difference in the form of Hamming distance between corresponding counter values.

6 Conclusion

This work presents a new cryptanalysis strategy called DINAMITE for a class of parallelisable ciphers using the counter mode, which combines IDs with GAD analysis to get a MITE scenario. A case study is presented in the form of a state (key) recovery attack on authenticated cipher PAEQ. The results improve upon the best known key-recovery attacks by a considerable margin using only a single ID trail. The main improvement lies in exploiting low inter-substate diffusion in the backward direction. First, a basic attack is proposed, which is improved incrementally exploiting more specific properties. The final time complexities stand at $2^{33}, 2^{48}, 2^{64}$ for PAEQ-64/80/128, respectively, while preserving data complexity of the previous best attack. Finally, a nonce-based differential attack is presented, which works with just two single-block plaintext/ciphertext pairs. As a future direction, it would be interesting to explore this idea for other ciphers, which meet the criteria for DINAMITE like NORX.

7 References

[1] Bellare, M., Namprempre, C.: 'Authenticated encryption: relations among notions and analysis of the generic composition paradigm'. Advances in Cryptology – ASIACRYPT 2000, 6th Int. Conf. on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, 3–7 December 2000, pp. 531–545

[2] Bellare, M., Namprempre, C.: 'Authenticated encryption: relations among notions and analysis of the generic composition paradigm', *J. Cryptol.*, 2008, **21**, (4), pp. 469–491

[3] Al Fardan, N.J., Paterson, K.G.: 'Lucky thirteen: breaking the TLS and DTLS record protocols'. IEEE Symp. on Security and Privacy 2013, Berkeley, CA, USA, 2013, pp. 526–540

[4] Duong, T., Rizzo, J.: 'Here come the XOR Ninjas'. White paper, Netifera, 2011

[5] CAESAR: 'Competition for authenticated encryption: security, applicability, and robustness', 2014. Available at http://competitions.cr.yp.to/caesar.html/

[6] Daemen, J., Rijmen, V.: 'The design of Rijndael: AES – the advanced encryption standard', '*Information security and cryptography*' (Springer, 2002)

[7] National Institute of Standards and Technology: 'SHA-3: cryptographic hash algorithm competition'. Available at http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

[8] Dinur, I., Dunkelman, O., Shamir, A.: 'Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials'. Report 2012/672, Cryptology ePrint Archive, 2012. Available at http://eprint.iacr.org/

[9] Peyrin, T.: 'Improved differential attacks for ECHO and Grøstl'. Advances in Cryptology – CRYPTO 2010, 30th Annual Cryptology Conf., Santa Barbara, CA, USA, 15–19 August 2010, pp. 370–392

[10] Boura, C., Chakraborti, A., Leurent, G., *et al.*: 'Key recovery attack against 2.5-round π-cipher'. 23rd Int. Workshop on Fast Software Encryption (FSE) 2016, Bochum, Germany, 20–23 March 2016, pp. 535–553

[11] Saha, D., Kakarla, S., Mandava, S., *et al.*: 'Gain: practical key-recovery attacks on round-reduced PAEQ'. 6th Int. Conf. on Security, Privacy, and Applied Cryptography Engineering (SPACE 2016), Hyderabad, India, 14–18 December 2016, pp. 194–210

[12] Saha, D., Kakarla, S., Mandava, S., *et al.*: 'Gain: practical key-recovery attacks on round-reduced PAEQ', *J. Hardware Syst. Secur.*, 2017, **1**, (3), pp. 282–296

[13] Biryukov, A., Khovratovich, D.: 'PAEQ v1', 2014

[14] Biryukov, A., Khovratovich, D.: 'PAEQ: parallelizable permutation- based authenticated encryption'. 17th Int. Conf. on Information Security (ISC) 2014, Hong Kong, China, 12–14 October 2014, pp. 72–89

[15] Saha, D., Chowdhury, D.R.: 'Encounter: on breaking the nonce barrier in differential fault analysis with a case-study on PAEQ'. 18th Int. Conf. on Cryptographic Hardware and Embedded Systems (CHES) 2016, Santa Barbara, CA, USA, 17–19 August 2016, pp. 581–601

[16] Bagheri, N., Mendel, F., Sasaki, Y.: 'Improved rebound attacks on AESQ: core permutation of CAESAR candidate PAEQ'. 21st Australasian Conf. on Information Security and Privacy (ACISP 2016), Melbourne, VIC, Australia, 4–6 July 2016, pp. 301–316

[17] Saha, D., Chowdhury, D.R.: 'Internal differential fault analysis of parallelizable ciphers in the counter-mode', *J. Cryptogr. Eng.*, 2017, available online

[18] Gligoroski, D., Mihajloska, H., Samardjiska, S., *et al.*: 'π-cipher v2.0'. Submission to the CAESAR competition, 2014. Available at http://competitions.cr.yp.to/caesar-submissions.html/

[19] Aumasson, J., Jovanovic, P., Neves, S.: 'NORX: parallel and scalable AEAD'. Computer Security – ESORICS 2014 – 19th European Symp. on Research in Computer Security, Wroclaw, Poland, 7–11 September 2014, pp. 19–36

[20] Aumasson, J., Jovanovic, P., Neves, S.: 'NORX v3.0', 2014

8 Appendix

8.1 Appendix 1: PPAE mode of operation

The details of the PPAE mode of operation are presented below in form of an Algorithm (see Fig. 2).

8.2 Appendix 2: DINAMITE attack on paeq-64

Here, we give a run of the DINAMITE attack on PAEQ 64-bit variant which has a time complexity of 2^{33}. It should be noted that though the data complexity is impractical (2^{89} blocks), the attack is verified by obtaining the two relevant branches required and working on them. Each table shows the execution of the phases of DINAMITE and is self-explanatory.

Nonce:
```
0x03 0x12 0x6d 0x2e 0x94 0x96 0x46 0xed
```
Key:
```
0x30 0×d4 0x87 0xff 0x1c 0x85 0xbb 0xab
```
Branch *i* plaintext block:
```
0xdb 0xec 0xaa 0xd1 0x59 0x58 0x31 0xc8
0x25 0xb6 0x6d 0xec 0x5a 0x18 0x88 0xd1
0x5e 0xde 0x95 0x5c 0x0f 0xd9 0x37 0x8f
0x7b 0x2b 0x4b 0xb3 0x1c 0x46 0xa1 0xbc
0x65 0x5d 0xda 0x38 0x4c 0xd7 0x55 0x56
0x0d 0xd5 0x39 0x92 0x22 0x8e 0x20 0x32
0xa1 0x31 0x5f 0x03 0x8f 0xb9
```
Branch *i* ciphertext block:
```
0xf2 0x9a 0xb4 0x07 0x8f 0x80 0x40 0x2a
0x50 0x79 0xb5 0xc5 0x38 0x1e 0x3f 0xdf
0x84 0xe2 0xf2 0xfd 0x9e 0x21 0x06 0x3c
0xc2 0xfe 0xec 0x85 0x5a 0x22 0x3b 0xd6
0xbb 0x79 0x56 0xe8 0xd2 0x85 0x1e 0x30
0x27 0x83 0xbe 0x08 0x5c 0x9e 0x7a 0x6f
0xe3 0x3c 0x5b 0x11 0xc1 0xd8
```
Branch *j* plaintext block:
```
0xdb 0xec 0xaa 0xd1 0x59 0x58 0x31 0xc8
0x25 0xb6 0x6d 0xec 0x5a 0x18 0x88 0xd1
0x5e 0xde 0x95 0x5c 0x0f 0xd9 0x37 0x8f
0x7b 0x2b 0x4b 0xb3 0x1c 0x46 0xa1 0xbc
0x65 0x5d 0xda 0x38 0x4c 0xd7 0x55 0x56
0x0d 0xd5 0x39 0x92 0x22 0x8e 0x20 0x32
0xa1 0x31 0x5f 0x03 0x8f 0xb9
```
Branch *j* ciphertext block:
```
0xd8 0xbe 0x2c 0xdd 0xfc 0x8c 0x58 0xd5
0xca 0x36 0xcc 0x19 0x4a 0x62 0x9f 0x39
0x94 0xa2 0x28 0x79 0x89 0x6b 0xa4 0x41
0x12 0xea 0x98 0xe2 0x05 0x37 0xad 0x5e
0x38 0xda 0x84 0xc3 0x73 0x4f 0x7e 0x6c
0xb8 0x02 0x5a 0xab 0xa1 0xb3 0x61 0x54
0x58 0xf9 0x21 0x99 0x72 0x50
```

8.2.1 ID phase: See Tables 7 and 8.

8.2.2 GAD and MITE: See Tables 9–11.

8.2.3 Final key recovery: See Table 12.

Table 7 ID using branches *i* and *j*

(a) $p \leftarrow D_0 \parallel i \parallel N \parallel \boldsymbol{X}$

40	00	00	00	00	00	00	00	00	00	00	00	03	94	**00**	**00**
40	00	00	00	00	00	00	00	00	00	00	00	12	96	**00**	**00**
01	00	00	00	00	00	00	00	00	00	00	00	6d	46	**00**	**00**
00	00	00	00	00	00	00	00	00	00	00	00	2e	ed	**00**	**00**

(b) $q \leftarrow D_0 \parallel j \parallel N \parallel \boldsymbol{X}$

40	00	3a	00	00	00	00	00	00	00	00	00	03	94	00	00
40	00	00	01	00	00	00	00	00	00	00	00	12	96	00	00
7e	00	00	00	00	00	00	00	00	00	00	00	6d	46	00	00
00	28	00	00	00	00	00	00	00	00	00	00	2e	ed	00	00

(c) $v \xrightarrow{\mathcal{S} \circ \mathrm{AESQ}^2} p$

ed	d5	aa	c2	d5	10	fa	aa	d5	e7	fa	aa	4b	d5	aa	fe
dc	ef	d0	76	ef	e9	82	d0	ef	1f	4e	d0	e1	ef	d0	30
3b	ef	d0	2e	ef	cb	5a	d0	ef	b0	42	d0	5f	ef	d0	dc
03	c7	86	67	c7	8f	4b	86	c7	58	5f	86	5f	c7	86	51

(d) $w \xrightarrow{\mathcal{S} \circ \mathrm{AESQ}^2} q$

ed	d5	aa	c2	d5	10	fa	aa	d5	e7	92	aa	4b	d5	aa	fe
dc	ef	d0	76	ef	e9	82	d0	ef	1f	7a	d0	e1	ef	d0	30
3b	ef	d0	2e	ef	cb	5a	d0	ef	b0	76	d0	5f	ef	d0	dc
03	c7	86	67	c7	8f	4b	86	c7	58	03	86	5f	c7	86	51

(e) $v_{c,*}^{m} \xrightarrow{\text{Assign guessed value}} g$

ed	d5	aa	c2	d5	10	fa	aa	d5	**bf**	fa	aa	4b	d5	aa	fe
dc	ef	d0	76	ef	e9	82	d0	ef	**5e**	4e	d0	e1	ef	d0	30
3b	ef	d0	2e	ef	cb	5a	d0	ef	**12**	42	d0	5f	ef	d0	dc
03	c7	86	67	c7	8f	4b	86	c7	**b7**	5f	86	5f	c7	86	51

(f) $w_{c,*}^{m} \xrightarrow{\text{Assign guessed value}} g$

ed	d5	aa	c2	d5	10	fa	aa	d5	**bf**	92	aa	4b	d5	aa	fe
dc	ef	d0	76	ef	e9	82	d0	ef	**5e**	7a	d0	e1	ef	d0	30
3b	ef	d0	2e	ef	cb	5a	d0	ef	**12**	76	d0	5f	ef	d0	dc
03	c7	86	67	c7	8f	4b	86	c7	**b7**	03	86	5f	c7	86	51

(g) $v \xrightarrow{\mu_5 \circ \rho_5 \circ \beta_5 \circ \mathcal{S} \circ \mathcal{R}_4 \circ \mathcal{R}_3} v$

05	70	de	70	3c	9d	36	ee	b0	17	20	84	a5	13	45	67
63	3d	c9	3a	07	f7	a5	1f	ba	4d	04	11	51	b7	64	b5
34	c1	63	03	5a	9e	e7	c4	63	14	fb	c3	30	31	b0	7b
18	e7	2d	6c	9b	f7	61	2e	6e	7b	2b	6e	b7	f2	f6	0b

(h) $w \xrightarrow{\mu_5 \circ \rho_5 \circ \beta_5 \circ \mathcal{S} \circ \mathcal{R}_4 \circ \mathcal{R}_3} w$

5b	68	48	df	bf	98	a6	c6	21	4a	90	1e	f6	2a	de	71
81	2d	82	95	84	f8	45	0b	2b	aa	2d	5c	a4	99	a4	a3
88	c9	28	e9	c4	94	97	d0	cb	ae	62	8e	96	26	70	41
46	ef	f0	29	86	f2	11	12	57	26	b2	b9	e4	e5	ad	27

(i) $f \leftarrow (v \oplus w)$															
5e	18	96	af	83	05	90	28	91	5d	b0	9a	53	39	9b	16
e2	10	4b	af	83	0f	e0	14	91	e7	29	4d	f5	2e	c0	16
bc	08	4b	ea	9e	0a	70	14	a8	ba	99	4d	a6	17	c0	3a
5e	08	dd	45	1d	05	70	3c	39	5d	99	d7	53	17	5b	2c

Table 8 Validation of column guess at ID phase

(a) $(P_i \oplus C_i)$															
XX	1e	71	d8	b7	67	31	a7	9a	8c	4b	87	5a	04	XX	XX
XX	d6	e2	29	0e	a1	b3	36	6a	d0	66	9a	5d	12	XX	XX
29	d6	75	62	da	91	b9	46	de	9e	2a	7e	42	4e	XX	XX
76	d8	cf	06	3c	f8	d5	64	24	52	56	10	0d	61	XX	XX

(b) $s \leftarrow \mathcal{S}^{-1}(P_i \oplus C_i)$															
d8	XX	4b	31	1e	04	9a	b7	71	XX	87	a7	XX	5a	8c	67
29	XX	66	b3	d6	12	6a	0e	e2	XX	9a	36	XX	5d	d0	a1
62	XX	2a	b9	d6	4e	de	da	75	XX	7e	46	29	42	9e	91
06	XX	56	d5	d8	61	24	3c	cf	XX	10	64	76	0d	52	f8

(c) $s \xrightarrow{(\alpha_5)^{-1} \circ (\mathcal{R}_6)^{-1} \circ \mathcal{S}^{-1} \circ (\mathcal{R}_7)^{-1} \circ (\mathcal{R}_8)^{-1}} s$															
XX	XX	XX	**81**	XX	XX	XX	**53**	XX	XX	XX	**4d**	XX	XX	XX	**d1**
51	XX	XX	XX	**c3**	XX	XX	XX	**80**	XX	XX	XX	**eb**	XX	XX	XX
XX	**57**	XX	XX	XX	**36**	XX	XX	XX	**40**	XX	XX	XX	**c9**	XX	XX
XX	XX	**2d**	XX	XX	XX	**f9**	XX	XX	XX	**2a**	XX	XX	XX	**28**	XX

(d) $t \leftarrow s \oplus f$															
XX	XX	XX	**2e**	XX	XX	XX	**7b**	XX	XX	XX	**d7**	XX	XX	XX	**c7**
b3	XX	XX	XX	**40**	XX	XX	XX	**11**	XX	XX	XX	**1e**	XX	XX	XX
XX	**5f**	XX	XX	XX	**3c**	XX	XX	XX	**fa**	XX	XX	XX	**de**	XX	XX
XX	XX	**f0**	XX	XX	XX	**89**	XX	XX	XX	**b3**	XX	XX	XX	**73**	XX

(e) $t \xrightarrow{\mathcal{R}_8 \circ \mathcal{R}_7 \circ \mathcal{S} \circ \mathcal{R}_6 \circ \alpha_5} t$															
XX	XX	XX	XX	**86**	**7e**	**0c**	**17**	XX	XX	XX	XX	XX	XX	XX	XX
XX	XX	XX	XX	**0c**	**9a**	**e2**	**e8**	XX	XX	XX	XX	XX	XX	XX	XX
XX	XX	XX	XX	**a5**	**fd**	**5d**	**ca**	XX	XX	XX	XX	XX	XX	XX	XX
XX	XX	XX	XX	**d4**	**e9**	**87**	**7c**	XX	XX	XX	XX	XX	XX	XX	XX

(f) $t' \leftarrow \mathcal{S}^{-1}(P_j \oplus C_j)$															
a1	XX	2b	93	86	7e	0c	17	69	XX	63	d3	XX	41	5e	bd
f5	XX	3a	ce	0c	9a	e2	e8	1d	XX	39	51	XX	66	fb	25
10	XX	b5	69	a5	fd	5d	ca	ef	XX	83	19	03	f9	3f	86
7a	XX	d7	c1	d4	e9	87	7c	80	XX	3d	71	52	c8	98	b2

Table 9 GAD and unsuccessful MITE

(a) $s \leftarrow \mathcal{S}^{-1}(P_i \oplus C_i)$															
d8	XX	4b	31	1e	04	9a	b7	71	XX	87	a7	XX	5a	8c	67
29	XX	66	b3	d6	12	6a	0e	e2	XX	9a	36	XX	5d	d0	a1
62	XX	2a	b9	d6	4e	de	da	75	XX	7e	46	29	42	9e	91
06	XX	56	d5	d8	61	24	3c	cf	XX	10	64	76	0d	52	f8

(b) $s \xrightarrow{\text{Guess unknown values in } s} s$

d8	**34**	4b	31	1e	04	9a	b7	71	**34**	87	a7	**34**	5a	8c	67
29	**db**	66	b3	d6	12	6a	0e	e2	**db**	9a	36	**db**	5d	d0	a1
62	**62**	2a	b9	d6	4e	de	da	75	**62**	7e	46	29	42	9e	91
06	**9c**	56	d5	d8	61	24	3c	cf	**9c**	10	64	76	0d	52	f8

(c) $s \xrightarrow{(\alpha_5)^{-1} \circ (\mathcal{R}_6)^{-1} \circ \mathcal{S}^{-1} \circ (\mathcal{R}_7)^{-1} \circ (\mathcal{R}_8)^{-1}} s$

41	6e	54	81	dc	b3	dc	53	4f	0e	c9	4d	65	f8	cb	d1
51	f6	b2	44	c3	2e	5f	df	80	75	e6	37	eb	48	ae	78
c0	57	ae	b9	3c	36	3a	3c	5b	40	13	f4	c0	c9	d4	15
dc	12	2d	fd	3f	1d	f9	40	3a	ec	2a	68	63	df	28	20

(d) $t \leftarrow s \oplus f$

1f	76	c2	2e	5f	b6	4c	7b	de	53	79	d7	36	c1	50	c7
b3	e6	f9	eb	40	21	bf	cb	11	92	cf	7a	1e	66	6e	6e
7c	5f	e5	53	a2	3c	4a	28	f3	fa	8a	b9	66	de	14	2f
82	1a	f0	b8	22	18	89	7c	03	b1	b3	bf	30	c8	73	0c

(e) $t \xrightarrow{\mathcal{R}_8 \circ \mathcal{R}_7 \circ \mathcal{S} \circ \mathcal{R}_6 \circ \alpha_5} t$ (no substate matches)

58	0e	42	78	86	7e	0c	17	5c	da	86	aa	4d	d4	83	96
f2	46	38	74	0c	9a	e2	e8	bb	b3	f6	ef	32	c2	fe	0b
c6	8d	e6	a1	a5	fd	5d	ca	e7	f2	92	47	37	c8	72	db
6a	34	83	da	d4	e9	87	7c	12	d6	6a	e9	2a	24	36	93

(f) $t' \leftarrow \mathcal{S}^{-1}(P_j \oplus C_j)$

a1	XX	2b	93	86	7e	0c	17	69	XX	63	d3	XX	41	5e	bd
f5	XX	3a	ce	0c	9a	e2	e8	1d	XX	39	51	XX	66	fb	25
10	XX	b5	69	a5	fd	5d	ca	Ef	XX	83	19	03	f9	3f	86
7a	XX	d7	c1	d4	e9	87	7c	80	XX	3d	71	52	c8	98	b2

Table 10 GAD and successful MITE (recovering substate 3)

(a) $s \leftarrow \mathcal{S}^{-1}(P_i \oplus C_i)$

d8	XX	4b	31	1e	04	9a	b7	71	XX	87	a7	XX	5a	8c	67
29	XX	66	b3	d6	12	6a	0e	e2	XX	9a	36	XX	5d	d0	a1
62	XX	2a	b9	d6	4e	de	da	75	XX	7e	46	29	42	9e	91
06	XX	56	d5	d8	61	24	3c	cf	XX	10	64	76	0d	52	f8

(b) $s \xrightarrow{\text{Guess unknown values in } s} s$

d8	**4d**	4b	31	1e	04	9a	b7	71	**4d**	87	a7	**4d**	5a	8c	67
29	**82**	66	b3	d6	12	6a	0e	e2	**82**	9a	36	**82**	5d	d0	a1
62	**1b**	2a	b9	d6	4e	de	da	75	**1b**	7e	46	29	42	9e	91
06	**32**	56	d5	d8	61	24	3c	cf	**32**	10	64	76	0d	52	f8

(c) $s \xrightarrow{(\alpha_5)^{-1} \circ (\mathcal{R}_6)^{-1} \circ \mathcal{S}^{-1} \circ (\mathcal{R}_7)^{-1} \circ (\mathcal{R}_8)^{-1}} s$

25	08	16	81	04	02	01	53	e0	17	05	4d	d4	71	74	d1
51	80	9d	05	c3	d1	2a	d9	80	c5	7c	2b	eb	f8	8d	0f
1c	57	9c	7a	a4	36	1f	12	cc	40	60	8e	5f	c9	f8	73
13	0b	2d	76	82	2f	f9	6c	22	17	2a	1d	74	b3	28	aa

(d) $t \leftarrow s \oplus f$														
7b	10	80	2e	87	07	91	7b	71	4a	b5	d7	87	48	ef	c7
b3	90	d6	aa	40	de	ca	cd	11	22	55	66	1e	d6	4d	19
a0	5f	d7	90	3a	3c	6f	06	64	fa	f9	c3	f9	de	38	49
4d	03	f0	33	9f	2a	89	50	1b	4a	b3	ca	27	a4	73	86

(e) $t \xrightarrow{\mathcal{R}_8 \circ \mathcal{R}_7 \circ \mathcal{S} \circ \mathcal{R}_6 \circ \alpha_5} t$														
a0	38	80	01	86	7e	0c	17	**69**	46	**63**	**d3**	c6	8a	23	42
05	97	af	f7	0c	9a	e2	e8	**1d**	91	**39**	**51**	a7	1d	09	a2
4c	9f	3b	0b	a5	fd	5d	ca	**ef**	3a	**83**	**19**	ed	3e	72	18
aa	3e	b9	7a	d4	e9	87	7c	**80**	7e	**3d**	**71**	b4	1d	c1	c8

(f) $t' \leftarrow \mathcal{S}^{-1}(P_j \oplus C_j)$															
a1	XX	2b	93	86	7e	0c	17	69	XX	63	d3	XX	41	5e	bd
f5	XX	3a	ce	0c	9a	e2	e8	1d	XX	39	51	XX	66	fb	25
10	XX	b5	69	a5	fd	5d	ca	ef	XX	83	19	03	f9	3f	86
7a	XX	d7	c1	d4	e9	87	7c	80	XX	3d	71	52	c8	98	b2

Table 11 GAD and successful MITE (recovering substate 4)

(a) $s \leftarrow \mathcal{S}^{-1}(P_i \oplus C_i)$															
d8	XX	4b	31	1e	04	9a	b7	71	4d	87	a7	XX	5a	8c	67
29	XX	66	b3	d6	12	6a	0e	e2	82	9a	36	XX	5d	d0	a1
62	XX	2a	b9	d6	4e	de	da	75	1b	7e	46	29	42	9e	91
06	XX	56	d5	d8	61	24	3c	cf	32	10	64	76	0d	52	f8

(b) $s \xrightarrow{\text{Guess unknown values in } s} s$															
d8	**6f**	4b	31	1e	04	9a	b7	71	4d	87	a7	**6f**	5a	8c	67
29	**86**	66	b3	d6	12	6a	0e	e2	82	9a	36	**86**	5d	d0	a1
62	**29**	2a	b9	d6	4e	de	da	75	1b	7e	46	29	42	9e	91
06	**76**	56	d5	d8	61	24	3c	cf	32	10	64	76	0d	52	f8

(c) $s \xrightarrow{(\alpha_5)^{-1} \circ (\mathcal{R}_6)^{-1} \circ \mathcal{S}^{-1} \circ (\mathcal{R}_7)^{-1} \circ (\mathcal{R}_8)^{-1}} s$															
0f	cf	16	81	f4	c5	01	53	3e	81	05	4d	e3	d4	74	d1
51	00	21	05	c3	5d	2a	d9	80	b8	83	2b	eb	3d	87	0f
1c	57	f3	39	a4	36	6d	b4	cc	40	54	ad	5f	c9	0f	55
f3	0b	2d	b9	62	2f	f9	cb	d0	17	2a	21	41	b3	28	59

(d) $t \leftarrow s \oplus f$															
51	d7	80	2e	77	c0	91	7b	af	dc	b5	d7	b0	ed	ef	c7
b3	10	6a	aa	40	52	ca	cd	11	5f	aa	66	1e	13	47	19
a0	5f	b8	d3	3a	3c	1d	a0	64	fa	cd	e0	f9	de	cf	6f
ad	03	f0	fc	7f	2a	89	f7	e9	4a	b3	f6	12	a4	73	75

(e) $t \xrightarrow{\mathcal{R}_8 \circ \mathcal{R}_7 \circ \mathcal{S} \circ \mathcal{R}_6 \circ \alpha_5} t$															
84	d7	88	36	86	7e	0c	17	69	46	63	d3	04	**41**	**5e**	**bd**
81	c6	6f	cd	0c	9a	e2	e8	1d	91	39	51	24	**66**	**fb**	**25**
f2	31	46	eb	a5	fd	5d	ca	ef	3a	83	19	**03**	**f9**	**3f**	**86**
ba	1f	86	1e	d4	e9	87	7c	80	7e	3d	71	**52**	**c8**	**98**	**b2**

(f) $t' \leftarrow \mathcal{S}^{-1}(P_j \oplus C_j)$														
a1	XX	2b	93	86	7e	0c	17	69	XX	63	d3	XX	41	5e	bd
f5	XX	3a	ce	0c	9a	e2	e8	1d	XX	39	51	XX	66	fb	25
10	XX	b5	69	a5	fd	5d	ca	ef	XX	83	19	03	f9	3f	86
7a	XX	d7	c1	d4	e9	87	7c	80	XX	3d	71	52	c8	98	b2

Table 12 Key recovery from recovered state of branch *j*

(a) Recovered state of branch *j* before the last permutation = *t*															
a1	28	2b	93	86	7e	0c	17	69	46	63	d3	04	41	5e	bd
f5	31	3a	ce	0c	9a	e2	e8	1d	91	39	51	24	66	fb	25
10	48	b5	69	a5	fd	5d	ca	ef	3a	83	19	03	f9	3f	86
7a	4b	d7	c1	d4	e9	87	7c	80	7e	3d	71	52	c8	98	b2

(b) Input of branch $j \xrightarrow{(\mathrm{AESQ}^8)^{-1} \circ \mathcal{S}} t$															
40	00	3a	00	00	00	00	00	00	00	00	00	03	94	**30**	**1c**
40	00	00	01	00	00	00	00	00	00	00	00	12	96	**d4**	**85**
7e	00	00	00	00	00	00	00	00	00	00	00	6d	46	**87**	**bb**
00	28	00	00	00	00	00	00	00	00	00	00	2e	ed	**ff**	**ab**