SCOPE: On the Side Channel Vulnerability of Releasing Unverified Plaintexts

Dhiman Saha(✉) and Dipanwita Roy Chowdhury

Crypto Research Lab, Department of Computer Science and Engineering,
IIT Kharagpur, Kharagpur, India
{dhimans,drc}@cse.iitkgp.ernet.in

Abstract. In Asiacrypt 2014, Andreeva et al. proposed an interesting idea of intermittently releasing plaintexts before verifying the tag which was inspired from various practical applications and constraints. In this work we try to asses the idea of releasing unverified plaintexts in the light of side channel attacks like fault attacks. In particular we show that this opens up new avenues of attacking the decryption module. We further show a case-study on the APE authenticated encryption scheme and reduce its key space from 2^{160} to 2^{50} using 12 faults and to 2^{24} using 16 faults on the decryption module. These results are of particular interest since attacking the decryption enables the attacker to completely bypass the nonce constraint imposed by the encryption. Finally, at the outset this work also addresses a related problem of fault attacks with partial state information.

Keywords: Authenticated encryption · Releasing unverified plaintexts · APE · Differential fault analysis

1 Introduction

In conventional security notions of Authenticated Encryption (AE), release of decrypted plaintext is subject to successful verification. In their pioneering paper in Asiacrypt 2014, Andreeva et al. challenged this model by introducing and formalizing the idea of *releasing unverified plaintexts* (RUP) [5,6]. The idea was motivated by a lot of practical problems faced by the classical approach like insufficient memory in constrained environments, real-time usage requirements and inefficiency issues. The basic idea is to separate the plaintext computation and verification during AE decryption, so that the plaintexts are always released irrespective of the status of the verification process. In order to assess the security under RUP and to bridge the gap with the classical approach, the authors have introduced two new definitions: INT-RUP (for integrity) and *plaintext awareness* or PA for privacy (in combination with IND-CPA).

In this work, we try to answer the question pertaining to RUP that arises from a side-channel view-point: *Can the ability to observe unverified plaintexts serve as a source of side-channel information?* Our research reveals that the answer is

O. Dunkelman and L. Keliher (Eds.): SAC 2015, LNCS 9566, pp. 417–438, 2016.
DOI: 10.1007/978-3-319-31301-6_24

affirmative with respect to differential fault analysis (DFA) [8,10–14,16] which is known to be one of the most effective side-channel attacks on symmetric-key constructions. The basic requirement of any form of fault analysis is the ability to induce a fault in the intermediate state of the cipher and consequently observe the faulty output. Our first observation is that in the classical approach where successful verification precedes release of plaintexts, fault attacks are infeasible. This is attributed to the fact that if the attacker induces a fault, the probability of the faulty plaintext to pass the verification is negligible, thereby denying the ability to observe the faulty output. This scenario changes in the presence of unverified plaintexts. So the first *scope* that RUP provides at the hands of the attacker is the ability to observe *faulty unverified plaintexts*. Our second observation is in terms of the *nonce* constraint. In Indocrypt 2014, Saha et al. studied the impact of the nonce constraint in their EscApe fault attack [15] on the authenticated cipher APE [3]. The authors showcased the restriction that the uniqueness of nonces imposes on the *replaying criterion*[1] of fault analysis and demonstrated the idea of *faulty collisions* to overcome it. In this work we argue that ability to attack the decryption, provided by RUP, gives the additional benefit of totally *bypassing* the nonce constraint. This follows from the very definition of AE decryption which allows an attacker to make multiple queries to the decryption oracle with the same nonce. Thus prospect of *nonce bypass* makes fault analysis highly feasible.

Following these observations, we mount Scope: a differential fault attack on the decryption of APE which is also one of the submissions to the on-going CAESAR [1] competition. The choice of APE is motivated by the fact that according to PA classification of schemes provided by Andreeva et al. in [5,6], *APE which has* ***offline decryption***, *is one of the CAESAR submissions that supports RUP*. Authenticated Permutation-based Encryption [4] or APE was introduced by Andreeva et al. in FSE 2014 and later reintroduced in CAESAR along with GIBBON and HANUMAN and an indigenous permutation called PRIMATE as part of the authenticated encryption family PRIMATEs [2,3]. We studied the fault diffusion in the inverse of the internal permutation PRIMATE of APE using random uni-word fault injections in the penultimate round. We capitalize on properties arising out of the non-square nature of the internal state and also the knowledge of the fault-free unverified plaintext. Our analysis shows average key-space reduction from 2^{160} to 2^{50} using 12 faults and to 2^{24} using 16 faults. Finally, this work identifies and addresses a broader problem in differential fault analysis: *Fault analysis with partial state information*. Since, only part of the state is observable, the fault analysis presented here deviates from the classical DFA [8,10–14,16] which generally assumes availability of the entire state at the output. Here we showcase that even knowledge of just one-fifth (the size of a plaintext block) of the state can be used to reconstruct the differential state and finally reduce the key-space. Moreover, close similarity between PRIMATE permutation and AES [9], automatically amplifies the scope

[1] The replaying criterion in differential fault analysis states that the attacker must be able to induce faults while replaying a previous fault free run of the algorithm.

of the results presented here. The contributions of this work can be summarized as below:

- Scrutinizing the recently introduced RUP model in the light of fault attacks.
- Showing that unverified plaintext can be an important source of side-channel information.
- Showing the feasibility of fault induction using nonce bypass.
- For the first time attacking the decryption of an AE scheme using DFA.
- Presenting SCOPE attack exploiting: fault diffusion in the last two rounds of the Inverse PRIMATE permutation and the ability to observe faulty unverified plaintexts.
- Finally, achieving a key space reduction from 2^{160} to 2^{50} with 12 faults and 2^{24} with 16 faults using the random word fault model.
- Moreover, this work also brings into focus the idea of fault analysis of AES based constructions with partial state information.

The rest of the work is organized as follows: Sect. 2 gives a brief description of the PRIMATE permutation and its inverse and introduces the notations used in this work. Section 3 looks at the RUP and classical models in the light of side-channel analysis. Some properties of APE decryption that become relevant in the presence of faults are discussed in Sect. 4. The proposed SCOPE attack is introduced in Sect. 5. Section 6 furnishes the experimental results with a brief discussion while Sect. 7 gives the concluding remarks.

2 Preliminaries

2.1 The Design of PRIMATE

PRIMATE has two variants in terms of size: PRIMATE-80 (200-bit permutation) and PRIMATE-120 (280-bit) which operate on states of (5×8) and (7×8) 5-bit elements respectively. The family consists of four permutations p_1, p_2, p_3, p_4 which differ in the round constants used and the number of rounds. All notations introduced in this section are with reference to PRIMATEs-80 with the APE mode of operation.

Definition 1 (Word). *Let $\mathbb{T} = \mathbb{F}[x]/(x^5 + x^2 + 1)$ be the field $\mathbb{F}_{2^5}$ used in the PRIMATE MixColumn operation. Then a **word** is defined as an element of $\mathbb{T}$.*

Definition 2 (State). *Let $\mathbb{S} = (\mathbb{T}^5)^8$ be the set of (5×8)-word matrices. Then the internal **state** of the PRIMATE-80 permutation family is defined as an element of $\mathbb{S}$. We denote a state $s \in \mathbb{S}$ with elements $s_{i,j}$ as $[s_{i,j}]_{5,8}$.*

$$s = [s_{i,j}]_{5,8}, \text{ where } \begin{cases} s_{i,j} \in \mathbb{T} \\ 0 \leq i \leq 4, \ 0 \leq j \leq 7 \end{cases} \tag{1}$$

In the rest of the paper, for simplicity, we omit the dimensions in $[s_{i,j}]_{5,8}$ and use $[s_{i,j}]$ as the default notation for the 5×8 state. We denote a column of $[s_{i,j}]$ as $s_{*,j}$ while a row is referred to as $s_{i,*}$. We now describe in brief the design of PRIMATE permutation. In this work we also deal with the inverse of the PRIMATE permutation. APE instantiates p_1 which is a composition of 12 round functions. The inverse permutation p_1^{-1} applies the round functions in the reverse order with each component operations itself being inverted. For the sake of consistency, in the rest of the work rounds of p^{-1} will be denoted w.r.t to the corresponding rounds of p. For instance, the last round of p^{-1} will be referred to as $\mathcal{R}_1^{-1}$ since functionally it is the inverse of the first round of p.

$$\begin{array}{l|l} p_1, p_1^{-1} : \mathbb{S} \longrightarrow \mathbb{S}, \quad p_1 = \mathcal{R}_{12} \circ \mathcal{R}_{11} \circ \cdots \circ \mathcal{R}_1 & p_1^{-1} = \mathcal{R}_1^{-1} \circ \mathcal{R}_2^{-1} \circ \cdots \circ \mathcal{R}_{12}^{-1} \\ \mathcal{R}_r, \mathcal{R}_r^{-1} : \mathbb{S} \longrightarrow \mathbb{S}, \quad \mathcal{R}_r = \alpha_r \circ \mu_r \circ \rho_r \circ \beta_r & \mathcal{R}_r^{-1} = \beta_r^{-1} \circ \rho_r^{-1} \circ \mu_r^{-1} \circ \alpha_r^{-1} \end{array}$$

where $\mathcal{R}_r$ is a composition of four bijective functions on $\mathbb{S}$ while $\mathcal{R}_r^{-1}$ denotes the inverse round function. The index r denotes the r^{th} round and may be dropped if the context is obvious. Here, the component function β represents the non-linear transformation SubBytes which constitutes word-wise substitution of the state according to predefined S-box. The definitions extend analogously for the inverse.

$$\beta_r, \beta_r^{-1} : \mathbb{S} \longrightarrow \mathbb{S}, \qquad s = [s_{i,j}] \overset{\beta}{\mapsto} [S(s_{i,j})], \qquad s = [s_{i,j}] \overset{\beta^{-1}}{\mapsto} [S^{-1}(s_{i,j})]$$

where $S : \mathbb{T} \longrightarrow \mathbb{T}$ is the S-box given in Table 1. The transformation ρ corresponds to ShiftRows which cyclically shifts each row of the state based on a set of offsets. The same applies to ρ^{-1} with only the direction of shift being reversed.

$$\rho_r, \rho^{-1} : \mathbb{S} \longrightarrow \mathbb{S}, \qquad s = [s_{i,j}] \overset{\rho}{\mapsto} [s_{i,(j-\sigma(i)) \bmod 8}], \qquad s = [s_{i,j}] \overset{\rho^{-1}}{\mapsto} [s_{i,(j+\sigma(i)) \bmod 8}]$$

where, $\sigma = \{0, 1, 2, 4, 7\}$ is the ShiftRow offset vector and $\sigma(i)$ defines shift-offset for the i^{th} row. The MixColumn operation, denoted by μ, operates on the state column-wise. μ is actually a left-multiplication by a 5×5 matrix (M_μ) over the finite field $\mathbb{T}$. For the InverseMixColumn (μ^{-1}), the multiplication is carried out using (M_μ^{-1}).

$$\mu_r : \mathbb{S} \longrightarrow \mathbb{S}, \qquad s = [s_{i,j}] \longmapsto s' = [s'_{i,j}], \qquad s'_{*,j} = M_\mu \times s_{*,j}$$

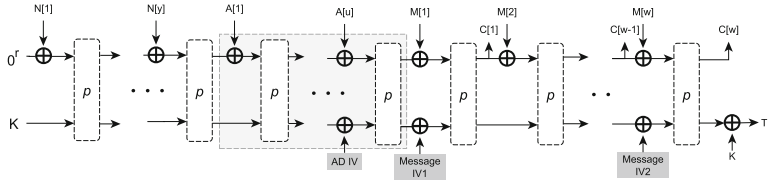
The last operation of the round function is α which corresponds to the round constant addition. The constants are the output $\{\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_{12}\}$ of a 5-bit LFSR and are xored to the word $s_{1,1}$ of the state $[s_{i,j}]$. α is involutory implying $\alpha = \alpha^{-1}$.

$$\alpha_r : \mathbb{S} \longrightarrow \mathbb{S}, \qquad [s_{i,j}] \longmapsto [s'_{i,j}], \qquad s'_{i,j} = \begin{cases} s_{i,j} \oplus \mathcal{B}_r \text{ if } i, j = 1 \\ s_{i,j}, \text{ Otherwise} \end{cases}$$

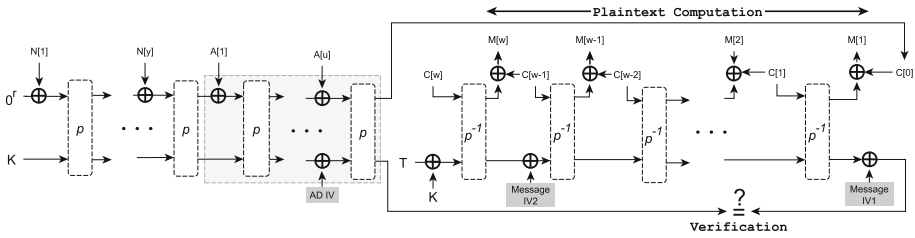
The APE mode of operation is depicted in Fig. 1. Here, $N[\cdot]$ represents a Nonce block while $A[\cdot]$ and $M[\cdot]$ denote blocks of associated data and message respectively. The IVs shown in Fig. 1 are predefined and vary according to the nature

Table 1. The PRIMATE 5-bit S-box

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$S(x)$	1	0	25	26	17	29	21	27	20	5	4	23	14	18	2	28
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$S(x)$	15	8	6	3	13	7	24	16	30	9	31	10	22	12	11	19



(a) APE Encryption



(b) APE Decryption

Fig. 1. The APE mode of operation

of the length of message and associated data. Figure 1a and b show the encryption and decryption modules of APE respectively. In is evident from Fig. 1b that the decryption starts from the last ciphertext block and proceeds in the reverse direction which implies that APE decryption is *offline*.

2.2 Notations

Definition 3 (Differential state). *A* ***differential state*** *is defined as the element-wise XOR between a state* $[s_{i,j}]$ *and the corresponding faulty state* $[s'_{i,j}]$.

$$s'_{i,j} = s_{i,j} \oplus \delta_{i,j}, \ \forall\, i, j \tag{2}$$

δ fully captures the initial fault as well as the dispersion of the fault in the state. In this work we assume induction of random faults in some word of a state. Thus, if the initial fault occurs in word $s_{I,J} \in s$, the differential state is of the following form:

$$\delta_{i,j} = \begin{cases} f : f \xleftarrow{R} \mathbb{T} \setminus \{0\}, \text{ if } (i = I, j = J) \\ 0, \text{ Otherwise} \end{cases} \tag{3}$$

If $\exists j : \delta_{i,j} = 0 \; \forall i$ then $\delta_{*,j}$ is called a *pure* column, otherwise $\delta_{*,j}$ is referred to as a *faulty* column.

Definition 4 (Hyper-column). *A **Hyper-column** is a* (5×1) *column vector where each element is again a vector of words i.e., a subset of* $\mathbb{T}$*. It is denoted by* $\mathcal{H}$*.*

$$\mathcal{H} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_4 \end{bmatrix} \quad \text{where } b_j \subset \mathbb{T}, \qquad \text{Also, } \mathcal{H} = \varnothing \quad \text{if} \quad \exists i : b_i = \varnothing$$

The Hyper-column helps to capture the candidate words for a column that result due to the fault analysis presented here. Also a hyper-column is considered to be empty if at least one of its component sets is empty.

Definition 5 (Hyper-state [15]). *A **Hyper-state** of a state* $s = [s_{i,j}]$*, denoted by* $s^h = [s^h_{i,j}]$*, is a two-dimensional matrix, where each element* $s^h_{i,j}$ *is a non-empty subset of* $\mathbb{T}$*, such that* s *is an element-wise member of* s^h*.*

$$s^h = \begin{bmatrix} s^h_{00} & s^h_{01} & \cdots & s^h_{07} \\ s^h_{10} & s^h_{11} & \cdots & s^h_{17} \\ \vdots & \vdots & \ddots & \vdots \\ s^h_{40} & s^h_{41} & \cdots & s^h_{47} \end{bmatrix} \quad \text{where } \begin{cases} s^h_{i,j} \subset \mathbb{T}, \;\; s^h_{i,j} \neq \varnothing \\ s_{i,j} \in s^h_{i,j} \; \forall i, j \end{cases} \tag{4}$$

The significance of a hyper-state s^h *is that the state* s *is in a way 'hidden' inside it. This means that if we create all possible states taking one word from each element of* s^h*, then one of them will be exactly equal to* s*.*

The hyper-state has some interesting properties with respect to the component transformations of the PRIMATE permutation and consequently its inverse. For instance all the inverse operations like InverseShiftRow(ρ^{-1}), InverseSubByte(β^{-1}), InverseAddRoundConstant(α^{-1}) can be applied on a hyper-state with little technical changes. This is possible since all these operations work word-wise and thus can be applied as a whole to each element-set of a hyper-state too with an equivalent effect. We define the analogs of these operations on a hyper-state as hyper-state-<operation>: $(\rho^{-1})', (\beta^{-1})', (\alpha_r^{-1})'$. The formal definitions are provided in Appendix A. Another observation of particular interest is that hyper-state-<operation>$(s^h) = (\text{<operation>}(s))^h$.

Definition 6 (Kernel [15]). *If s^h is a hyper-state of s, then **Kernel** of a column $s^h_{*,j} \in s^h$, denoted by $\mathcal{K}^{s^h_{*,j}}$, is defined as the cross-product of $s^h_{0,j}, s^h_{1,j}, \cdots, s^h_{4,j}$.*

$$\mathcal{K}^{s^h_{*,j}} = \left\{ w_k : w_k^T \in \bigtimes_{i=0}^{4} s^h_{i,j},\ 1 \leq k \leq \prod_{i=0}^{4} |s^h_{i,j}| \right\}$$

Subsequently, Kernel of the entire hyper-state is the set of the Kernels of all of its columns: $\mathcal{K}^{s^h} = \{\mathcal{K}^{s^h_{*,0}}, \mathcal{K}^{s^h_{*,1}}, \cdots, \mathcal{K}^{s^h_{*,7}}\}$

Here, w_k^T represents the transpose of w_k, thereby implying that w_k is a column vector. One should note that $s_{*,j} \in \mathcal{K}^{s^h_{*,j}}\ \forall j$. Thus each column of s is contained in each element of $\mathcal{K}^{s^h}$. We now define an operation $(\mu^{-1})'$ over the Kernel of a hyper-state which is equivalent to μ^{-1} that operates on a state.

Definition 7 (Kernel-InverseMixColumn).
*The **Kernel-InverseMixColumn** transformation denoted by $(\mu^{-1})'$ is the left multiplication of M_μ^{-1} to each element of each $\mathcal{K}^{s^h_{*,j}} \in \mathcal{K}^{s^h}$.*

$$(\mu^{-1})'(\mathcal{K}^{s^h_{*,j}}) = \{M_\mu^{-1} \times w_i,\ \forall w_i \in \mathcal{K}^{s^h_{*,j}}\}$$
$$(\mu^{-1})'(\mathcal{K}^{s^h}) = \{(\mu^{-1})''(\mathcal{K}^{s^h_{*,0}}), (\mu^{-1})'(\mathcal{K}^{s^h_{*,1}}), \cdots, (\mu^{-1})'(\mathcal{K}^{s^h_{*,7}})\}$$

An important implication is that $(\mu^{-1})'(\mathcal{K}^{s^h}) = \mathcal{K}^{(\mu^{-1}(s))^h}$. The notion of **Hyper-state** and **Kernel** will be used in the OUTBOUND phase of SCOPE detailed in Subsect. 5.3.

3 RUP in the Light of Side-Channels

RUP which has been argued to be a very desirable property can be a major source for side channel information. In this work we try to study how RUP stands out in the light of fault attacks. Our research reveals that RUP opens up an exploitable opportunity with respect to fault analysis which would not have been possible if verification would precede release of the plaintexts. Moreover, attacking the decryption also allows the attacker to bypass the nonce constraint imposed by the encryption. It has been shown that nonce based encryption has an automatic protection against DFA and hence ability to bypass the nonce constraint exposes the AE scheme to fault attacks. In the rest of the paper we refer to the classical model that does not allow RUP as **RVP (Release of Verified Plaintexts)**. We now argue why RVP has an implicit protection against fault attacks which makes attacking the decryption infeasible.

In order to understand the significance of the *scope* that the RUP model puts at the hands of the attacker, one has to be aware of why fault analysis in the RVP model is infeasible. According to the classical RVP model, the decryption oracle returns the entire plaintext if the verification passes and $\perp$ otherwise.

Now we define the term *faulty forgery* as the ability of the attacker to produce a plaintext ($p* \neq p$) after inducing faults such that the verification passes. Now, in standard fault analysis it is assumed that *the attacker can induce random faults in the state but the value of the fault is unknown.* Under this scenario, the probability of the attacker to produce a faulty forgery is negligible (Fig. 2).

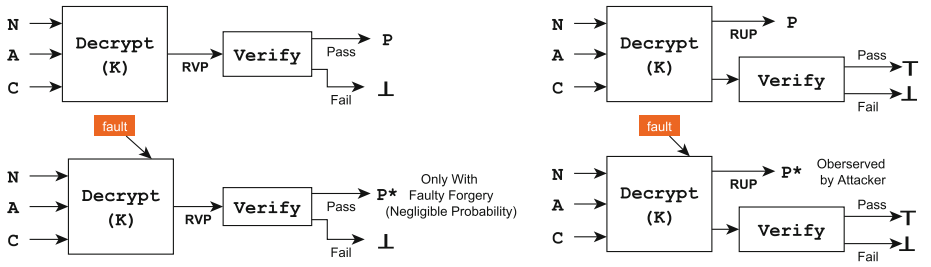


Fig. 2. RVP vs RUP from the perspective of fault analysis

On the contrary RUP gives the attacker the scope of inducing random faults while decrypting any chosen or known ciphertext and **unconditionally** observe the corresponding faulty plaintexts (which would never have passed verification in the RVP model). This power opens up the side-channel for fault analysis and is the basis of the differential fault attack presented in this work. Moreover, the ability to attack the decryption has the additional and important advantage of bypassing the nonce constraint that is imposed while making encryption queries. This magnifies the feasibility of mounting fault attacks.

In the next section, we look at some of the features of APE decryption and the inverse PRIMATE permutation p^{-1} that gain importance from a fault attack perspective. Finally, building upon these observations we introduce the SCOPE attack where for the first time we show how the decryption can also be attacked under RUP to retrieve the entire internal state of p^{-1} leading to recovery of the key with practical complexities.

4 Analyzing APE Decryption in the Presence of Faults

In this section we look at certain properties of APE decryption that become relevant in the context of RUP and from the prospect of fault induction. We first look at a property which by itself is of no threat to the security of APE but becomes exploitable in the presence of faults in the RUP scenario.

4.1 The Block Inversion Property

The Block Inversion Property is purely attributed to the APE mode of operation. This property allows the attacker to retrieve partial information about the contents of the state matrix after the last round InverseMixColumn operation.

Property 1. *If the state after μ_1^{-1} in $\mathcal{R}_1^{-1}$ (the last round of p^{-1}) be represented as $t = [t_{i,j}]$ and the released plaintext block and next ciphertext block be p and c respectively, then $t_{0,*}$ is public by the following expression:*

$$t_{0,i} = S(v_i) \quad \textit{where,} \begin{cases} v_i \in (p \oplus c), \\ S \rightarrow \texttt{PRIMATE Sbox (Table 1)} \end{cases}$$

Analysis: By virtue of the APE mode of operation and the SPONGE [7] construction it follows, the *rate* part (top row of the state) after $\mathcal{R}_1^{-1}$ of p^{-1} is released after XORing with the next ciphertext block as the plaintext block (which can be observed unconditionally under RUP). If the state after $\mathcal{R}_1^{-1}$ be $s = [s_{i,j}]$ then $p \oplus c$ gives back $s_{0,*}$. We can now invert this block to get inside $\mathcal{R}_1^{-1}$ despite partial knowledge of the state. This becomes possible since β operates word-wise and ρ operates row-wise. Moreover, ρ can be ignored for it has no effect on top row as the shift-offset is zero. Thus applying β on $s_{0,*}$ we get the value of $t_{0,*}$. However, the inversion stops here since μ operates column-wise and only word of each column is known. ■

Later in this work we show how the SCOPE attack can exploit the Block Inversion Property along with RUP and use both faulty and fault-free plaintexts to reconstruct differential state after μ_2^{-1} in $\mathcal{R}_2^{-1}$. We now study the fault induction and diffusion in the state of p^{-1} which is vital to understanding of the attack presented here.

4.2 Fault Diffusion in the Inverse PRIMATE Permutation

In this section we describe the induction and diffusion of faults in the inverse (p^{-1}) of the PRIMATE permutation during APE decryption. In fact, our intention is to study the fault diffusion in the *differential state* of p^{-1} which we exploit to formulate a fault attack on APE Decryption. The fault induction and subsequent differential plaintext block formation are illustrated in Fig. 3. One can see from Fig. 3 that the fault is induced in the input of the penultimate round $\mathcal{R}_2^{-1}$ of p^{-1}. The logic behind this will be clear from the following important property of fault diffusion in the internal state of p^{-1}.

Property 2. *If a single column is faulty at the start of $\mathcal{R}_{r+1}^{-1}$ then there are exactly three fault-free words in each row of the differential state after $\mathcal{R}_r^{-1}$.*

Analysis: This property surfaces because in two rounds the fault does not spread to the entire state matrix. This is primarily attributed to the fact that the state matrix is non-square. To visualize this we need to first look at fault diffusion in the $\mathcal{R}_{r+1}^{-1}$ round. Let us denote the differential state at the input of $\mathcal{R}_{r+1}^{-1}$ as $s = [s_{i,j}]$. This analysis takes into account the structural dispersion of the fault and is independent of the actual value of s. At the beginning of $\mathcal{R}_{r+1}^{-1}$ only one column $s_{*,j}$ is faulty. The operation α^{-1} is omitted from analysis since round-constant addition has no effect on the differential state.

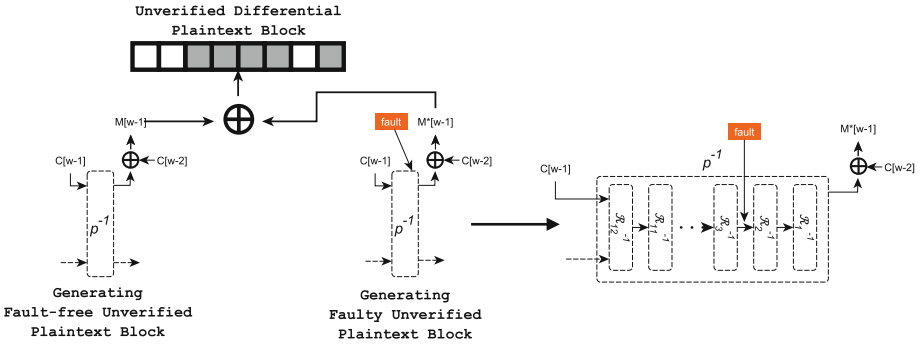


Fig. 3. Fault induction in APE decryption before releasing unverified plaintext and the unverified differential plaintext block

– Fault diffusion in $\mathcal{R}_{r+1}^{-1}$
  - μ_{r+1}^{-1} : Intra-column diffusion. Fault spreads to entire column $s_{*,j}$.
  - ρ_{r+1}^{-1} : No diffusion, fault shifts to the words $\{s_{i,(j+\sigma(i)) \bmod 8} : 0 \leq i < |\sigma|\}$.
  - β_{r+1}^{-1} : No diffusion, fault limited to the same words as after ρ_{r+1}^{-1}.

$$s_{*,j} \xrightarrow{\mu_{r+1}^{-1}} s_{*,j} \xrightarrow{\beta_{r+1}^{-1} \circ \rho_{r+1}^{-1}} \{s_{i,(j+\sigma(i)) \bmod 8}\} \tag{5}$$

– Fault diffusion in $\mathcal{R}_r$
  - μ_r^{-1} : Fault spreads to each column $s_{*,(j+\sigma(i)) \bmod 8}$.
  - ρ_r^{-1} : No diffusion, fault shifts to the words $\{s_{i,(j+\sigma(i)+\sigma(k)) \bmod 8} : 0 \leq i,k < |\sigma|\}$.
  - β_r^{-1} : No diffusion, fault limited to the same words as after ρ_r^{-1}.

$$\{s_{i,(j+\sigma(i)) \bmod 8}\} \xrightarrow{\mu_r^{-1}} s_{*,(j+\sigma(i)) \bmod 8} \xrightarrow{\beta_r^{-1} \circ \rho_r^{-1}} \{s_{i,(j+\sigma(i)+\sigma(k)) \bmod 8}\} \tag{6}$$

From (5) and (6) we have the following relation between the faulty column $s_{*,j}$ at the start $\mathcal{R}_{r+1}^{-1}$ and the faulty words after $\mathcal{R}_r^{-1}$.

$$s_{*,j} \xrightarrow{\mathcal{R}_r^{-1} \circ \mathcal{R}_{r-1}^{-1}} \{s_{i,(j+\sigma(i)+\sigma(k)) \bmod 8} : 0 \leq i,k < |\sigma|\} \tag{7}$$

For PRIMATE-80, $\sigma = \{0, 1, 2, 4, 7\}$, implying that $|\sigma| = 5$. From (7), we have $|\{s_{i,(j+\sigma(i)+\sigma(k)) \bmod 8}\}| = 25$. Thus a single faulty column before $\mathcal{R}_{r+1}^{-1}$ results in 25 faulty words at the end of $\mathcal{R}_r^{-1}$. Moreover, for each value of i we have $|\{s_{i,(j+\sigma(i)+\sigma(k)) \bmod 8} : 0 \leq k < 5\}| = 5$ implying that each row has 5 faulty words and respectively $8 - 5 = 3$ fault-free words at the end of $\mathcal{R}_r^{-1}$. An illustration of the above analysis with the source fault in column $s_{*,3}$ is depicted in Fig. 4. ■

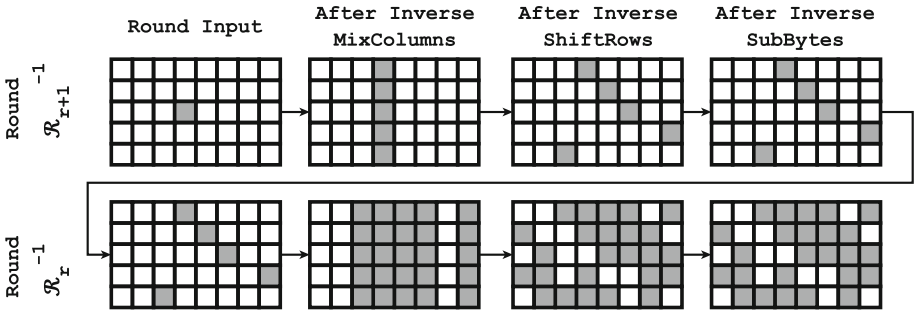


Fig. 4. 2-round fault diffusion with a uni-word fault in column $s_{*,3}$

4.3 The Bijection Lemma

This lemma stems out of the property mentioned above and is pivotal in increasing the efficiency of the SCOPE attack. Again it is a direct consequence of the non-square nature of the internal state of p^{-1}.

Lemma 1. *If fault is induced in the j^{th} column of the state at the input of $\mathcal{R}_{r+1}^{-1}$, then the fault-free words in the differential plaintext block released after $\mathcal{R}_{r}^{-1}$ are $((j+3),(j+5),(j+6))$ mod 8.*

Proof. This directly follows from relation (7). One can recall that for APE decryption under RUP, the first row of the state is released after XORing with next ciphertext block. However, since we are considering a differential here, the effect of the ciphertext block is nullified. Now, for $i = 0$, from relation (7) we have $\{s_{0,(j+\sigma(0)+\sigma(k)) \bmod 8} : 0 \leq k < 5\} = \{s_{0,j}, s_{0,j+1}, s_{0,j+2}, s_{0,j+4}, s_{0,j+7}\}$ which signifies the set of faulty words in the differential plaintext block. Hence, the complement of this set w.r.t the set of all the words in the plaintext block is $\{s_{0,j+3}, s_{0,j+5}, s_{0,j+6}\}$, which signify the fault-free words. ■

The implication of this lemma is that there exists a bijection between the positions of the fault-free words in the differential plaintext block released after $\mathcal{R}_{r}^{-1}$ and position of the column in which the fault was induced before $\mathcal{R}_{r+1}^{-1}$. This is vital to the analysis presented in this work and shows that by looking at the unverified differential plaintext block the attacker can ascertain the column position of the fault. This makes the attack 8 times faster. However, this information is not sufficient to guess the row position since all faults in the same column will produce the same pattern for the fault-free words.

In case of $p^{-1}, r = 1$ and the Bijection Lemma implies that by looking at the unverified differential block (Fig. 3) released after $\mathcal{R}_{1}^{-1}$, the attacker can ascertain in which column the fault was induced before $\mathcal{R}_{2}^{-1}$. With knowledge of all these characteristics of the APE mode of operation as well as p^{-1}, we are now in a place to finally introduce the differential fault attack developed in this work: SCOPE.

5 SCOPE: Differential Fault Analysis of APE Decryption (Exploiting Release of Unverified Plaintexts)

The first task is to run APE decryption and observe the released unverified plaintexts. Next the attacker queries the decryption with same set of inputs. Recall, that *nonce constraint can be bypassed* by definition. Every time, while replaying the decryption, he induces a random uni-word fault at the input of $\mathcal{R}_2^{-1}$ of p^{-1} during the processing of the same ciphertext block. *By RUP principle, the attacker can observe the corresponding faulty plaintext blocks.* The fault-free plaintext block (p) along with each corresponding faulty plaintexts block (p'_i) are stored. Now using the Bijection Lemma every differential plaintext block ($p \oplus p'_i$) is analyzed to get the faulty column before $\mathcal{R}_2^{-1}$. The information is stored in the fault count vector ($\mathcal{F}$) which is an array keeping count of the number of faults traced back to each column before $\mathcal{R}_2^{-1}$. For each unverified faulty plaintext, the INBOUND phase is initiated to get back a set of hyper-columns. The process is detailed in the next subsection.

5.1 The INBOUND Phase

The main aim of this phase is to reduce the number of candidate words for the column to which the fault was traced back. Let the state after μ_1^{-1} for the fault-free case be $s = [s_{i,j}]$ and for the faulty case be $s' = [s'_{i,j}]$. Now, by virtue of the Block Inversion Property, $s_{0,*}$ and $s'_{0,*}$ are known to the attacker. He now exploits the relation between the differential state before and after μ_1^{-1} that arises from the fault diffusion to reconstruct the entire differential state after $\mathcal{R}_1^{-1}$. To be more precise, the attacker is interested in the nature of the differential block ($s_{0,*} \oplus s'_{0,*}$). Due to the InverseMixColumn operation every non-zero word ($s_{0,*} \oplus s'_{0,*}$) is a multiple of the non-zero word in the corresponding column before μ_1^{-1} and the relation is governed by the InverseMixColumns matrix. Thus if the source fault is in column 4, ($s_{0,*} \oplus s'_{0,*}$) is of the following form: $\{0, 0, x_1 \times F_5, x_2 \times F_1, x_3 \times F_2, x_4 \times F_3, 0, x_5 \times F_4\}$. Now to get back each F_i from the differential row, the attacker makes use of the Factor Matrix given in Table 2. As one can notice the Factor Matrix is a circulant matrix. The i^{th} row corresponds to the factors to be used if the source fault is in the i^{th} column. The '*' represents the positions of the zero values of the corresponding differential row. So, the attacker retrieves each F_i by word-wise Galois Field division of the differential row ($s_{0,*} \oplus s'_{0,*}$) by using the appropriate row from the Factor Matrix. The method of generating the Factor Matrix is detailed in Appendix D.

The attacker now has the entire differential state after $\mathcal{R}_2^{-1}$. He cannot invert further deterministically since β is nonlinear. However, as ρ and β are commutative, he can apply ρ before β. By virtue of the fault diffusion described in Property (2), the differential state after β_2^{-1} has only one non-zero column and it is the same column where the fault was induced. The attacker now solves differential equations involving the same column at the input of β_2^{-1} which arise due the InverseMixColumns of $\mathcal{R}_2^{-1}$. However, these equations are characterized by

Table 2. The Factor Matrix

6	22	31	*	1	*	*	15
15	6	22	31	*	1	*	*
*	15	6	22	31	*	1	*
*	*	15	6	22	31	*	1
1	*	*	15	6	22	31	*
*	1	*	*	15	6	22	31
31	*	1	*	*	15	6	22
22	31	*	1	*	*	15	6

the row in which the initial fault was induced. One can recall that from Lemma 1, that the information available is not sufficient to ascertain the exact row. For instance, the fault invariants[2] for different rows of column 4 is shown in Fig. 5. So, the attacker solves the five sets of equations assuming all the possibilities. Out of these one set corresponds to the actual row that was affected. Solving the equations results in significant reduction in column space. The candidate words that satisfy the equations are stored into hyper-columns (Definition 4). So each row guess results in a different hyper-column and hence there can be maximum of 5 hyper-columns. However, one may encounter a lot of wrong candidate words getting accepted as they satisfy the wrong set of equations arising from the incorrect row guess. We refer to all accepted words other than the legitimate ones as NOISE. Thus one run of the INBOUND Phase returns a set of hyper-columns with a maximum cardinality of 5. The phase, is repeated for each faulty unverified plaintext and corresponding set of hyper-columns appropriately stored in a set of sets of hyper-columns: $\mathbb{H}$. After all faulty plaintexts have been processed, the set $\mathbb{H}$ along with the fault count vector $\mathcal{F}$ are passed on to the NOISE handling phase.

5.2 NOISE Handling

Here the attacker takes the advantage of the fact that while he induces random uni-word faults in input of $\mathcal{R}_2^{-1}$, there is a high probability that some faults get induced in the same column. Thus he will have multiple sets of hyper-columns from the INBOUND phase that reduced the column space for the same column before $\mathcal{R}_2^{-1}$. On the contrary, it might so happen that only one fault gets induced for a particular column. The worst-case scenario occurs if none of the induced faults affects some specific column. The former cases are dealt with in the next subsections while for the later case the attacker is left with exhaustive search implying that NOISE handling phase will return a hyper-column that spans the entire column space.

[2] A discussion on the generation and nature of the fault invariants is furnished in Appendix C.

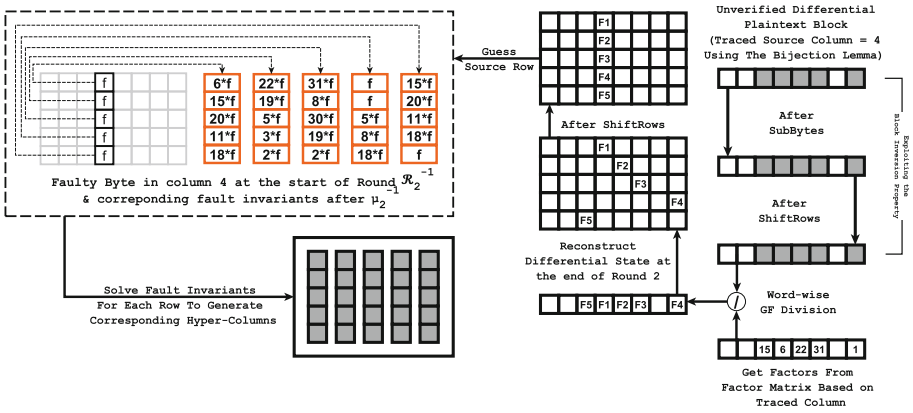
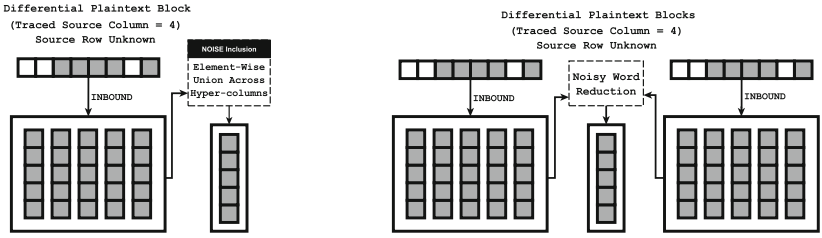


Fig. 5. Generation of hyper-columns using a word-fault at the beginning of $\mathcal{R}_2^{-1}$



(a) Noise Inclusion if Only One Fault Traced back to a Column

(b) Noise Reduction if Multiple Faults Traced back to the same Column

Fig. 6. The Noise handling phase

Noise **Inclusion.** When the attacker traces only one fault back to a column, he faces an ambiguity regarding the source row. In this scenario, he has no other option but to include all the hyper-columns for the next phase of the attack. So he includes all the Noise in the final step. So Noise Inclusion corresponds to word-wise union of all hyper-columns as depicted in Fig. 6a. Noise Inclusion, definitely, increases the column-space, however, computer simulations show that the final cardinality is still much better that brute force.

Noise **Reduction.** When the attacker traces multiple faults to the same column, he can significantly reduce the column space by eliminating Noisy hyper-columns. For e.g. if two faults are traced back to column x, then the attacker has two sets of hyper-columns. He now takes the cross-product of these two sets. Every element of the cross-product is a pair of hyper-columns. He now takes the set intersection between each such pair. The result is again a hyper-column with the cardinality of its component sets highly reduced. However, if the

hyper-column turns out to be empty[3], it is discarded. Experiments show that most of the elements from the cross-product get eliminated due to this and the attacker is left with a single final hyper-column. In case multiple hyper-columns remain, a element-wise union is taken to form the final hyper-column.

This NOISE handling phase is repeated for all the columns and returns a set of eight hyper-columns for the last phase of the attack.

procedure HANDLENOISE($\mathcal{F}, \mathbb{H}$) ▷ $\mathcal{F} \rightarrow$ Fault count vector; $\mathbb{H} \rightarrow$ Set of all sets of hyper-columns
  $\mathcal{H}_U = \{b_0, b_1, b_2, b_3, b_4\}^T$, where $b_i = \{0, 1, \cdots, 31\}$ ▷ $\mathcal{H}_U \rightarrow$ Exhaustive Hyper-column
  for $i = 0 : 7$ **do**
    if $\mathcal{F}(i) = 0$ **then** ▷ If no fault traced to column i
      $\mathcal{H}_i = \mathcal{H}_U$ ▷ Set hyper-column to be exhaustive
    else if $\mathcal{F}(i) = 1$ **then** ▷ If only one fault traced back to column i: NOISE Inclusion
      $\mathcal{H}_i = \bigcup \mathbb{H}_{i,1}$ ▷ Take union over the hyper-column set
    else if $\mathcal{F}(i) > 1$ **then** ▷ If multiple faults traced back to column i: NOISE Reduction
      $\mathcal{C} = \mathbb{H}_{i,1} \times \mathbb{H}_{i,2} \times \cdots \times \mathbb{H}_{i,\mathcal{F}(i)}$ ▷ Take cross-product over all sets of hyper-columns
      $j = 0$
      for $\forall d \in \mathcal{C}$ **do** ▷ Each element $d \in \mathcal{C}$ is a set of hyper-columns and $|d| = \mathcal{F}(i)$
        $\mathcal{H}^j_{temp} = \bigcap d$ ▷ Take intersection over each set of hyper-columns
        if $\mathcal{H}^j_{temp} \neq \varnothing$ **then** ▷ Recall (Definition 4) $\mathcal{H} = \varnothing$ if $\exists i : b_i = \varnothing, b_i \in \mathcal{H}$
          $j = j + 1$
        end if
      end for
      $\mathcal{H}_i = \bigcup \mathcal{H}^j_{temp}$ ▷ Take union over the set of all non-empty hyper-columns
    end if
  end for
  return $\{\mathcal{H}_0, \mathcal{H}_1, \cdots, \mathcal{H}_7\}$
    ▷ Each $\mathcal{H}_i$ has candidate words for the i^{th} column in the state after μ_2^{-1}.
end procedure

[3] Recall, that by Definition 4, a hyper-column is empty if any of its components is empty.

5.3 The Outbound Phase

The Outbound phase of Scope is inspired from the Outbound phase of the EscApe [15] attack proposed by Saha et al. in Indocrypt 2014 and closely follows it. It borrows the idea of a *Hyper-state* and *Kernel* from there. The input to this phase is the set of eight hyper-columns. Since none of the hyper-columns are empty, they can easily be combined structurally to form the hyper-state of the state after μ_2^{-1}. Let us denote the state by $s = [s_{i,i}]$ and then the hyper-state is s^h. This hyper-state s^h captures the reduced state-space for the state s that has been generated using the last two phases. In this phase we want to further reduce the state-space using knowledge of the fault-free plaintext block by again employing the Block Inversion property. This phase is called Outbound since it tries to move outward from μ_2^{-1}. We start by propagating further into $\mathcal{R}_2^{-1}$ and then move into $\mathcal{R}_1^{-1}$ by applying some hyper-state-<operations> on s^h. The steps of the Outbound phase are enlisted below.

1. The attacker starts the Outbound phase by applying Hyper-state InverseShiftRow transformation (Definition 8) on s^h followed by Hyper-state InverseSubByte (Definition 9) on s^h. This completes $\mathcal{R}_2^{-1}$ propagation.

$$s^h \xrightarrow{(\rho^{-1})'} (\rho_2^{-1}(s))^h \xrightarrow{(\beta^{-1})'} (\beta_2^{-1}(\rho_2^{-1}(s)))^h \rightarrow v^h(say)$$

2. We now move forward into the last round of $p^{-1} : \mathcal{R}_1^{-1}$. Let us denote the state $\beta_2^{-1}(\rho_2^{-1}(s))$ as v. We now apply Hyper-state InverseAddRoundConstant (Definition 10): $(\alpha_1^{-1})'$ on the hyper-state v^h. The next step is to compute the Kernel for $(\alpha_1^{-1}(v))^h : \mathcal{K}^{(\alpha_1^{-1}(v))^h}$.

$$v^h \xrightarrow{(\alpha_1^{-1})'} (\alpha_1^{-1}(v))^h \xrightarrow{\text{Compute Kernel}} \mathcal{K}^{(\alpha_1^{-1}(v))^h}$$

3. Then the attacker applies the Kernel-InverseMixColumn transformation on the Kernel $\mathcal{K}^{(\alpha_1^{-1}(v))^h}$

$$\mathcal{K}^{(\alpha_1^{-1}(v))^h} \xrightarrow{(\mu^{-1})'} \mathcal{K}^{(\mu_1^{-1}(\alpha_1^{-1}(v)))^h}$$

4. Next comes the reduction step. It can be noted that $\mathcal{K}^{(\mu_1^{-1}(\alpha_1^{-1}(v)))^h}$ represents the kernel for the hyper-state of $(\mu_1^{-1}(\alpha_1^{-1}(v)))$. i.e., the state just before the application of ρ_1^{-1}. Now let $t = (\mu_1^{-1}(\alpha_1^{-1}(v)))$. Then by the Block Inversion property, the actual value of $t_{0,*}$ is known. This knowledge is used to reduce the size of each $\mathcal{K}^{t^h_{*,j}} \in \mathcal{K}^{t^h}$. This reduction algorithm is almost similar to ReduceKernel given in [15] and is restated in Appendix B for easy reference.

A pictorial description of the Outbound phase is furnished in Fig. 7. Thus, after the Outbound phase we get a reduced Kernel for the state at the end of μ_1^{-1}. Every element of the cross-product of Kernels of each column is a candidate state. Finally, applying ρ_1^{-1} and β_1^{-1} on each candidate state produces

the reduced state-space at the end of $\mathcal{R}_1^{-1}$ of p^{-1}. This reduced state-space directly corresponds to the key-space of the state since recovering the internal state implies recovery of the key. The overall SCOPE attack is summarized by the following algorithm:

procedure SCOPE$(p, c, \{p'_1, p'_2, \cdots, p'_n\})$

$$\left\{\begin{array}{c} p \rightarrow \text{Fault-free unverified plaintext} \\ c \rightarrow \text{Next ciphertext block} \\ p'_i \rightarrow \text{Faulty unverified plaintext} \\ n \rightarrow \#\text{ of faulty outputs} \end{array}\right\}$$

  for $i = 0 : 7$ **do**
    $\mathcal{F}(i) = 0$ ▷ Initialize fault count vector
  end for
  for $i = 1 : n$ **do**
    $col \xleftarrow{\text{Lemma (1)}} (p \oplus p'_i)$ ▷ Get faulty column using the Bijection Lemma
    $\mathcal{F}(col) = \mathcal{F}(col) + 1$ ▷ Update fault count vector
    $\mathbb{H}_{col,\mathcal{F}(col)} \leftarrow$ INBOUND$\big(p, c, p'_i\big)$ ▷ Get set of hyper-columns for column col (Fig 5)
  end for
  $\{\mathcal{H}_0, \mathcal{H}_1, \cdots, \mathcal{H}_7\} \leftarrow$ HANDLENOISE$\big(\mathcal{F}, \mathbb{H}\big)$ ▷ Final set of 8 hyper-columns
  $s^h \leftarrow \{\mathcal{H}_0, \mathcal{H}_1, \cdots, \mathcal{H}_7\}$ ▷ Construct Hyper-State
  $\mathcal{K}_{red}^{t^h} \leftarrow$OUTBOUND$(s^h, p \oplus c)$ ▷ Get Reduced Kernel using the Block Inversion Property (Fig. 7)
  $\mathbb{S} = \varnothing$ ▷ Initialze final state-space set
  for all $w \in \left(\bigtimes_{j=0}^{7} \mathcal{K}_{red}^{t^h_{*,j}}\right)$ **do** ▷ Unroll Kernel to generate state-space
    $\mathbf{s} = \beta_1^{-1}(\rho_1^{-1}(w))$
    $\mathbb{S} = \mathbb{S} \cup \{\mathbf{s}\}$
  end for
  return $\mathbb{S}$ ▷ Return final state-space after $\mathcal{R}_1^{-1}$
end procedure

6 Experimental Results and Discussion

SCOPE was verified by extensive computer simulations. The experimental results confirm large scale reduction in the state-space and consequently the key-space. Average case analysis reveals that with 12 random uni-word faults at the input of $\mathcal{R}_2^{-1}$, the state-space at the end of $\mathcal{R}_1^{-1}$ reduces from 2^{160} to 2^{50} while 16 faults give a reduced state-space of 2^{24}. It is interesting to note that the fault distribution had a direct impact on state(key)-space reduction. To highlight the impact we look at two different fault distributions with 12 faults. Let the fault count vectors be $\mathcal{F}_1 = \{1, 2, 3, 0, 2, 2, 1, 1\}$ and $\mathcal{F}_2 = \{2, 2, 2, 0, 2, 2, 1, 1\}$.

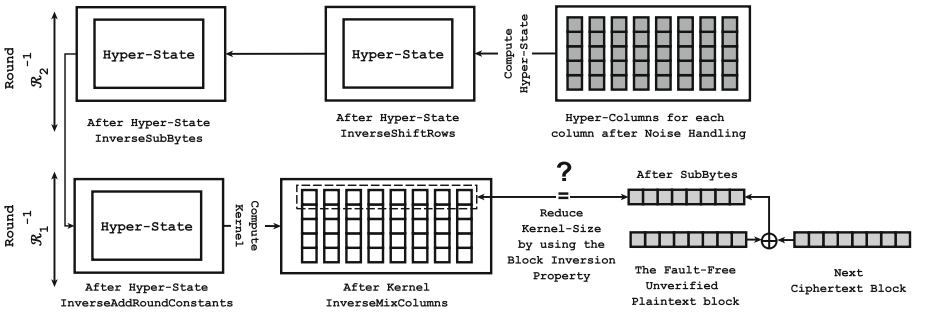


Fig. 7. Final reduction in state-space using fault-free unverified plaintext block

The average reduction with these distributions are 2^{45} and 2^{28} respectively. This extreme variance in the reduced key-spaces is attributed firstly to the fact that $\mathcal{F}_2$ is a more uniform distribution. Secondly, $\mathcal{F}_1$ has three columns which get just one fault. Thus, NOISE reduction cannot be applied to them. While for $\mathcal{F}_2$ such cases are two which leads to a better NOISE reduction in the NOISE handling phase and hence the better reduction in overall key-space. To conclude, it can be said that best results are obtained when fault distribution is such that maximum number of columns receive at least two faults.

It might be argued that in comparison to ESCAPE attack by Saha et al. SCOPE requires more faults. However, it must be kept in mind that SCOPE works with only partial state information while ESCAPE has the full state at its disposal. Moreover, since SCOPE attacks APE decryption it can bypass the nonce constraint and hence also avoid the need of *faulty collisions* which are inevitable for ESCAPE. Overall, SCOPE shows an interesting case-study where an AES-like construction is analyzed using faults with *partial* state information available to the attacker.

7 Conclusion

In this work we explore the *scope* provided by the RUP model with regards to fault analysis. We argue that ability to observe unverified plaintext opens up the fault side channel to attackers which is otherwise unavailable or available with negligible probability. In this work for the first time we show how the decryption of APE, an AE scheme that supports RUP, becomes vulnerable to DFA. Experiments reveal that using the random word fault model the key-space can be reduced from 2^{160} to 2^{50} using 12 faults while 16 faults reduce it to 2^{24}. An important implication of the ability to attack the decryption using RUP is that the attacker can totally bypass the nonce constraint imposed by the encryption. Finally, this work shows that though RUP is a desirable property addressing a lot of practical problems, it provides a unique scope to the attacker for mounting the SCOPE fault attack.

A Some More Definitions

Definition 8 (Hyper-state InverseShiftRow). *This transformation, denoted by* $(\rho^{-1})'$*, corresponds to cyclically* ***right*** *shifting each row of* s^h *based on the predefined set of offsets* σ*.*

$$(\rho^{-1})' : \bigtimes_{j=0}^{7} s^h_{i,j} \longrightarrow \bigtimes_{j=0}^{7} s^h_{i,(j+\sigma(i)) \bmod 8} \ \forall i$$

$$s^h_{i,*} \longmapsto s^h_{i,(*+\sigma(i)) \bmod 8} \ \forall i$$

It is interesting to note that, every word in the state $\rho^{-1}(s)$ will be a member of the corresponding element of $(\rho^{-1})'(s^h)$, thereby implying that $(\rho^{-1})'(s^h) = (\rho^{-1}(s))^h$.

Definition 9 (Hyper-state InverseSubByte). *This transformation, denoted by* $(\beta^{-1})'$*, corresponds to word-wise substitution of each word of each element-set of* s^h *based on the inverse of the PRIMATE Sbox.*

$$(\beta^{-1})' : w \longmapsto S^{-1}(w), \quad \forall w \in s^h_{i,j}, \quad \forall i, j$$

Definition 10 (Hyper-state InverseAddRoundConstant). *This transformation, denoted by* $(\alpha_r^{-1})'$*, corresponds to appropriate round constant addition to all words of element-set* $s^h_{1,1}$ *of the hyper-state* s^h*.*

$$(\alpha_r^{-1})' : w \longmapsto w \oplus \mathcal{B}_r, \quad \forall w \in s^h_{1,1}$$

B REDUCEKERNEL Algorithm [15]

procedure REDUCEKERNEL($\mathcal{K}^{t^h}, t_{0,*}$)
  for $j = 0 : 7$ **do**
    for all $\{e_0, e_1, e_2, e_3, e_4\}^T \in \mathcal{K}^{t^h_{*,j}}$ **do**
      if $e_0 \neq t_{0,j}$ **then**
        $\mathcal{K}^{t^h_{*,j}} = \mathcal{K}^{t^h_{*,j}} - \{e_0, e_1, e_2, e_3, e_4\}^T$
      end if
    end for
  end for
  $\mathcal{K}^{t^h}_{red} = \mathcal{K}^{t^h}$
  return $\mathcal{K}^{t^h}_{red}$
end procedure

C A Discussion on the Fault Invariants After μ_2^{-1} in $\mathcal{R}_2^{-1}$

The formation of the fault invariants due to a uni-word fault in the input of $\mathcal{R}_2^{-1}$ is completely governed by the InverseMixColumn Matrix (M_μ^{-1}). It is worth

mentioning that the fault-invariants are unique w.r.t to the row in which the fault was induced in the state before $\mathcal{R}_2^{-1}$ and are independent of the column. This implies that fault-invariants are fixed for a particular row irrespective of which column the faulty word belonged to. Let the differential state before μ_2^{-1} be $s = [s_{i,j}]$ and the one after μ_2^{-1} be $t = [t_{i,j}]$. Also let the differential fault value be f. Figure 8 depicts an example of fault invariant formation in $t_{*,7}$ for a word-fault in $s_{4,7}$. It can be noted that **value** of the invariant given in $t_{*,7}$ is same for any word-fault in $s_{4,*}$ i.e. the fifth row of s. However, the position of the invariant is due the column position of the fault which is the eighth column of s. Table 3 gives the exhaustive list of fault invariants exploited in SCOPE and their relation to the row location of the induced word fault.

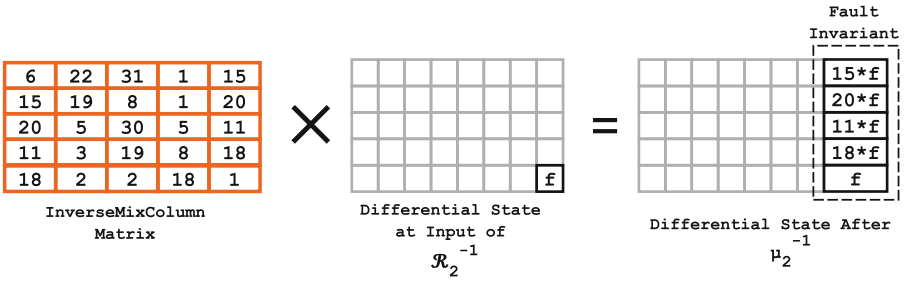


Fig. 8. Formation of fault invariant. Value of the fault invariant determined by M_μ^{-1} and the **row** position of the word-fault before $\mathcal{R}_2^{-1}$. Position of fault invariant determined by **column** position of the word-fault.Formation of fault invariant. Value of the fault invariant determined by M_μ^{-1} and the **row** position of the word-fault before $\mathcal{R}_2^{-1}$. Position of fault invariant determined by **column** position of the word-fault.

Table 3. Fault invariants exploited in SCOPE attack and the rows they correspond to.

Row position of fault	0	1	2	3	4
Fault invariant value	$\begin{bmatrix} 6\times f \\ 15\times f \\ 20\times f \\ 11\times f \\ 18\times f \end{bmatrix}$	$\begin{bmatrix} 22\times f \\ 19\times f \\ 5\times f \\ 3\times f \\ 2\times f \end{bmatrix}$	$\begin{bmatrix} 31\times f \\ 8\times f \\ 30\times f \\ 19\times f \\ 2\times f \end{bmatrix}$	$\begin{bmatrix} f \\ f \\ 5\times f \\ 8\times f \\ 18\times f \end{bmatrix}$	$\begin{bmatrix} 15\times f \\ 20\times f \\ 11\times f \\ 18\times f \\ f \end{bmatrix}$

D Constructing the Factor State

The factor state as stated earlier gives us the factors with which the differential row computed using the Block Inversion property needs to be divided to

reconstruct the state at the end of $\mathcal{R}_2^{-1}$. The pseudo-code for generating the factor matrix is given below:

procedure GENERATEFACTORMATRIX
  Take state $s = [s_{i,j}]$
  for $col = 0 : 7$ **do**
    $s_{i,j} = 0,\ \forall\ i, j$
    $s_{*,col} = [1, 1, 1, 1, 1]^T$
    $s = \mu^{-1}(\rho^{-1}(s))$
    $\texttt{factorMat}_{col,*} = s_{1,*}$
  end for
  return `factorMat`
end procedure

References

1. CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. http://competitions.cr.yp.to/caesar.html
2. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Wang, Q., Yasuda, K.: PRIMATEs v1 (2014). http://competitions.cr.yp.to/round1/primatesv1.pdf
3. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Wang, Q., Yasuda, K.: PRIMATEs v1.01 (2014). http://primates.ae/wp-content/uploads/primatesv1.01.pdf
4. Andreeva, E., Bilgin, B., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: APE: Authenticated Permutation-Based Encryption for Lightweight Cryptography. In: Cid, C., Rechberger, C. (eds.) FSE 2014. LNCS, vol. 8540, pp. 168–186. Springer, Heidelberg (2015). https://lirias.kuleuven.be/handle/123456789/450105
5. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to Securely Release Unverified Plaintext in Authenticated Encryption. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8873, pp. 105–125. Springer, Heidelberg (2014)
6. Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Mouha, N., Yasuda, K.: How to securely release unverified plaintext in authenticated encryption. Cryptology ePrint Archive, Report 2014/144 (2014). http://eprint.iacr.org/
7. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Cryptographic sponge functions. http://sponge.noekeon.org/CSF-0.1.pdf
8. Biham, E., Shamir, A.: Differential Fault Analysis of Secret Key Cryptosystems. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 513–525. Springer, Heidelberg (1997)
9. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer, Heidelberg (2002)
10. Dusart, P., Letourneux, G., Vivolo, O.: Differential Fault Analysis on A.E.S. IACR Cryptology ePrint Archive 2003, 10 (2003). http://eprint.iacr.org/2003/010
11. Giraud, C.: DFA on AES. In: Dobbertin, H., Rijmen, V., Sowa, A. (eds.) AES 2005. LNCS, vol. 3373, pp. 27–41. Springer, Heidelberg (2005)

12. Moradi, A., Shalmani, M.T.M., Salmasizadeh, M.: A Generalized Method of Differential Fault Attack Against AES Cryptosystem. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 91–100. Springer, Heidelberg (2006)
13. Mukhopadhyay, D.: An Improved Fault Based Attack of the Advanced Encryption Standard. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 421–434. Springer, Heidelberg (2009)
14. Piret, G., Quisquater, J.-J.: A Differential Fault Attack Technique Against SPN Structures, with Application to the AES and KHAZAD. In: Walter, C.D., Koç, Ç.K., Paar, C. (eds.) CHES 2003. LNCS, vol. 2779, pp. 77–88. Springer, Heidelberg (2003)
15. Saha, D., Kuila, S., Chowdhury, D.R.: EscApe: Diagonal Fault Analysis of APE. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014. LNCS, vol. 8885, pp. 197–216. Springer, Heidelberg (2014)
16. Saha, D., Mukhopadhyay, D., RoyChowdhury, D.: A Diagonal Fault Attack on the Advanced Encryption Standard. Cryptology ePrint Archive, Report 2009/581 (2009). http://eprint.iacr.org/