On Misuse of Nonce-Misuse Resistance

Adapting Differential Fault Attacks on (few) CAESAR Winners

Mustafa Khairallah
SPMS, Nanyang Technological University
Singapore
mustafam001@e.ntu.edu.sg

Shivam Bhasin
Temasek Labs @ NTU
Singapore
sbhasin@ntu.edu.sg

Anupam Chattopadhyay
SCSE, Nanyang Technological University
Singapore
anupam@ntu.edu.sg

***Abstract*—New cryptographic schemes are often built upon old and proven primitives which withstood long public scrutiny. The recently concluded CAESAR competition saw several authenticated ciphers which were directly built upon proven primitives like AES. However, any attacks associated with these underlying primitives become a vulnerability to the whole scheme. AES, which is considered theoretically secure, has a very low fault resistance against differential fault attacks (DFA) requiring only 1-2 faults. In this paper, we study DFA attacks on some of the CAESAR competition winners with AES block cipher as the underlying primitive. We study the challenges imposed by the design of these modes, such as masking of the cipher-text. We also show that a very small number of nonce repetition and faults is required to extend the original attack on AES, which makes it very practical. We show that OCB and COLM need 1 nonce repetition and 3 faults only to uniquely identify the Key.**

I. Introduction

Authenticated Encryption with Associated Data (AEAD) has become one of the most important primitives in Symmetric Key Cryptography, a status that is highlighted by the recently concluded CAESAR competition [1] and the new NIST Lightweight Cryptography competition [2]. In February 2019, the final portfolio of the CAESAR competition winners was announced, consisting of 6 algorithms divided into three use-cases:

1) Lightweight Applications: Ascon [7] and ACORN [12].
2) High Speed Applications: OCB [11] and AEGIS [13].
3) Defense-in-Depth Applications: Deoxys [9] and COLM [3].

While these operation modes use underlying primitives that are known to be insecure against physical attacks, such as Differential Fault Attacks (DFA [6]), it is not always clear how these attacks can be extended to the modes using such primitives. For example, OCB, COLM and Deoxys all use the AES SPN as an underlying primitive[1]. However, due to the mode design, the attacker may not have the same level of access to the primitive as in the case of the stand-alone AES cipher.

In this paper we investigate the challenges that the mode imposes on the attacker and how the attacker can bypass them in order to apply the DFA attack on the underlying primitive and recover the key, which we consider to be the main goal of DFA. These challenges can be due to how the primitive is used inside the mode or due to the security model of the nonce usage. The latter case is less relevant to the practical setting of the DFA. In fact it is considered near-impossible to prevent nonce *misuse* in the presence of physical attacks [5], which lead to the emergence of the *misuse-resilience* [4], [5], as opposed to *misuse-resistance*, to capture attackers that can force the device to reuse the nonces even for a temporary period of time. In this model the attacker is allowed to reuse the nonce during the attack phase. However, he cannot reuse the nonce, afterwards. In our analysis, we use this definition (Section II) to attack modes in the first two use-cases. The two winners in the Defense-in-Depth use-case are secure against nonce-misuse, so such distinction is not required. However, since our DFA analysis targets the recovery of the master key, the two definitions are equivalent, as the once the key is recovered the attacker can generate cipher-texts using fresh nonces.

The rest of the paper is organised as follows. Section II recalls background concepts on nonce isuse resistance/resilience and DFA on SPN. Section V proposes the adaption of DFA on AES to AES-OCB/COLM-128, two of the CAESAR winners. Finally conclusions are drawn in Section VI.

II. Background

A. Misuse-resistance vs. Misuse Resilience

Most nonce-based AEAD modes are secure only in the nonce-respecting model, i.e. the attackers advantage is negligible only if the nonce is never repeated. However, it is not always clear what is the effect of repeating the nonce for the same message as opposed to repeating the nonce for different messages. For example, while a nonce-respecting mode can be broken if the attacker can generate $E_K(N, M_1)$ and $E_K(N, M_2)$, where $M_1 \neq M_2$, usually the case when $M_1 = M_2$ only leaks the fact that $M_1 = M_2$, leading to a trivial distinguishing attack, which does not necessarily affect the security beyond this specific attack. To avoid such trivial attacks in security analyses, adversaries are assumed to never ask for the encryption of exactly the same query more than once.

However, the impact of nonce misuse on the security is not the same for all nonce-respecting modes. For example, if an attacker can repeat the nonce a few times while executing OCB,

[1] Deoxys uses Deoxys-BC, which uses the AES SPN with different key schedule and more rounds. the DFA on AES and Deoxys-BC will be almost exactly the same.

he can completely break the security of the system. However, in modes like GCM, even if the attacker can repeat the nonce, he can only infer information on messages encrypted with exactly the same nonce, with very little impact on messages encrypted with other nonces. Hence, the definition of misuse-resilient modes emerged.

Definition 1: A nonce-based AEAD mode of operation is misuse resilient if the information obtained by reusing a certain nonce N_a during the challenge queries does not give him an advantage with regards to distinguishing messages encrypted under a fresh nonce N_f [4].

Since most nonce-based AEAD schemes assume the nonce-respecting model, these modes are insecure even against misuse resilience, with few exceptions as mentioned earlier. However, while the nonce-respecting model is sound in the black-box security model, it is not practical in the presence of a malicious physical adversary who can induce faults in to the circuit, software. On the other hand, if we assume the adversary can fully control the nonce, breaking such modes can become trivial. Hence, for the sake of DFA analysis, we propose a weakened adversary. Some fault attacks, such as Statistical Fault Attacks (SFA) and Statistical Ineffective Fault Attacks (SIFA) assume that the adversary has no control over the nonce at all. Yet, such attacks can still be used to break the system using a large number of faults (> 100 faults). We view such approach as too conservative towards the other end of the spectrum, as if the attacker has the capabilities to inject faults into the cipher, he can also inject faults into the nonce circuitry, forcing repetitions. In order to reach a more moderate view, we assume the attacker can cause the nonce to repeat only once.

B. DFA on SPNs

In [10], Khairallah et al. studied the general structure of DFA attacks on SPNs and provided an efficient method to analyze a wide range of SPNs with low computational complexity. The analysis used a new tool called Joint Difference Distribution Table (JDDT), which is characteristic for the cipher and can be computed once, independent of the key. For example, the space of a 32-bit word of the last round key of AES can be reduced from 2^{32} to $\sim 2^8$ keys with a single byte-random fault, with computational complexity of 2^8 32-bit XORs, as opposed to 2^{32} partial decryption using previous analyses methods. In this paper, while analyzing AES based modes, we use the analysis and algorithm from [10] as a black box.

C. The Joint Difference Distribution Table (JDDT)

An SPN can be defined as a group of non-linear functions $S(x)$, where $S(x)$ consists of a linear part (diffusion layer) and a non-linear part (Sbox). The JDDT utilises the observation that when a single word difference Δ is injected into the input of the diffusion layer in round j, the inputs to n corresponding Sboxes in round $j+1$ are not independent, but are $\{l_1(\Delta), l_2(\Delta), ...l_n(\Delta)\}$, where $l_i(x)$ is a linear function from 1 word to 1 word, which corresponds to the difference propagation through the diffusion layer. Hence, instead of analyzing each of the Sboxes in round $j+1$ independently, we can analyze them using their joint differential properties, expressed by the JDDT, which is actually a portion of the DDT of the function $[A_1, A_2, ...A_n] \leftarrow S(\{a_1, a_2, ...a_n\})$. The JDDT of n Sboxes consists of exactly $2^{-(n-1)b}$ rows of the overall DDT of $S(x)$. The purpose of the JDDT is to provide candidates for the output value of $S(x)$, given Δ and the output difference.

We compute the JDDT as follows: we consider all the 2^{4b} possible output differences and divide them into four b-bit differences. We use each of these values to access the corresponding DDT and find all the possible input differences corresponding to this value. Typically, for good Sboxes, these would be four lists of around 2^{b-1} values each, which means $2^{4(b-1)}$ possible values for the difference at the output of the diffusion layer. However, only a subset of these values satisfies the relation $\{l_1(\Delta), l_2(\Delta), l_3(\Delta), l_4(\Delta)\}$. Hence, they are tested and only the solutions corresponding to valid differences are stored into the JDDT.

D. Single Fault Attack on SPNs

In [10], the authors generalized an attack against a wide class of SPNs that minimizes the number of faults required. The attack consists of two phases. In the offline phase, the JDDT of the cipher is calculated and the optimal fault locations are identified. In the online phase, the fault is injected in the identified location and propagates as shown in Figure 1. Once the output difference of a group is observed, the JDDT is accessed, and based on the assumptions on the input difference values (fault model, Sbox/Mixing differential properties, etc), a set of potential output values is required. These values are XORed with the ciphertext to get a set of key-words candidates of this group.

In [10], the authors provided the expected complexity and number of faults for attacking several SPN-based ciphers, including AES. It was shown that AES using a single fault, the key space can be reduced to 2^8 keys using random-byte faults and less than two keys using a known-byte fault.

III. Applying DFA on Deoxys

Deoxys-II [9] is a nonce-misuse resistant AEAD mode. It was selected as the first choice for high security applications in the CAESAR competition. It uses the Deoxys-BC as an underlying tweakable block cipher (TBC), and it has two variants; the first one with 128-bit key and the second one with 256-bit key, both of them use 120-bit nonce and 128-bit tag. The attack we describe here targets only the first variant. However, it can easily be extended to the other variant by attacking more rounds.

The encryption part of Deoxys-II is shown in Figure 2 from [9]. The tag variable depends on both the message and the associated data. However, it is a public parameter as it is released by the encryption algorithm. The attacker can choose a 2-block message and ask for its encryption with a given nonce. Since the attacker knows the message, as this is a

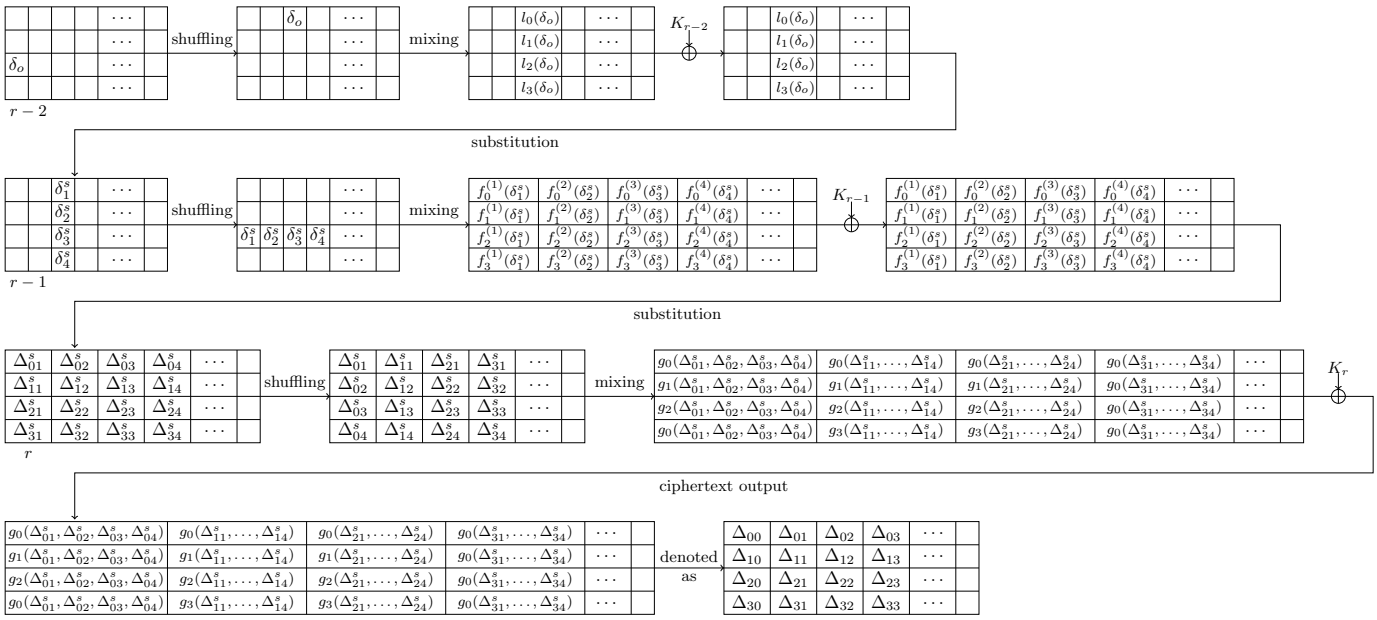

Fig. 1: 3 Round DFA on SPNs [10]

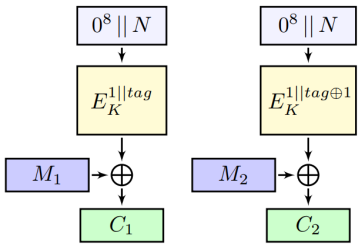

Fig. 2: The encryption part of Deoxys-II for a two-block message

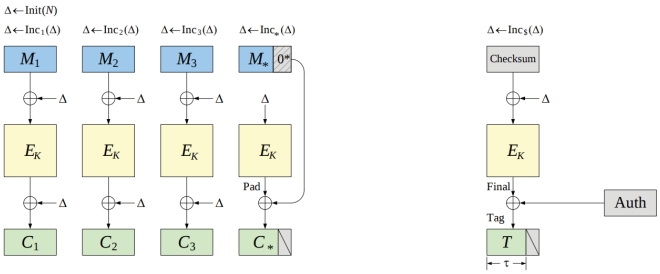

Fig. 3: The Encryption part of the OCB AEAD mode [11]

chosen plaintext attack, he can access the output of Deoxys-BC, applying the DFA against AES on both blocks, recovering 2^{32} candidates for the last round key for each encryption call. However, unlike AES, the round keys in Deoxys-BC are a function of both the master key and the tweak input, $1|tag$ and $1|tag \oplus 1$ in this case. In order to recover the master key, we need to remove the impact of the tweak input from the last round key. In order to do so, we use the fact that Deoxys-BC uses the Tweakey framework [8], where the key schedule operates on each part of the tweakey independently. Hence, the attacker can regenerate all the contributions of the tweak input to the round keys, converting the last round key candidates to master key candidates. Finally, the attacker can find the intersection between the two sets of key candidates, identifying the master key.

IV. Applying DFA on OCB

OCB is one of the winners of the CAESAR competition for High Performance Applications. The designers propose several variants using different AES variants and key sizes. The attack inthis section targets the variants based on AES-128. The attack can be extended to other variants using different variants of DFA on AES. Figure 3 from [11] shows the encryption part of the OCB mode. It can be seen that most encryptions are hard to attack using DFA due to the masking of the input and output with the variable Δ. However, we notice that we can overcome this challenge by carefully choosing the plaintext used for the attack. If $|M| = 120$ and $|AD| = 0$, then the encryption algorithm will access the block cipher only twice:

1) $C = M \oplus \texttt{truncate}_{120}(\mathsf{ENC}_K(\Delta))$.
2) $T = \mathsf{ENC}(M|10^7 \oplus \Delta)$.

The second encryption can be targeted with standard DFA against AES. This leads to 2^{32} key candidates with complexity 2^8 or 2^8 key candidates with complexity 2^{30}. The first encryption can also be attacked. However, the attacker needs to guess 8 bits of the ciphertext output difference, since the ciphertext will be truncated to only 120 bits. This increases the number of key candidates resulting for this part of the attack to 2^40 instead of 2^32. The probability of intersection between the two sets of key candidates is either $2^{72}/2^{128} = 2^{-56}$ or $2^{48}/2^{128} = 2^{-80}$. Consequently, 2 faults are enough to uniquely

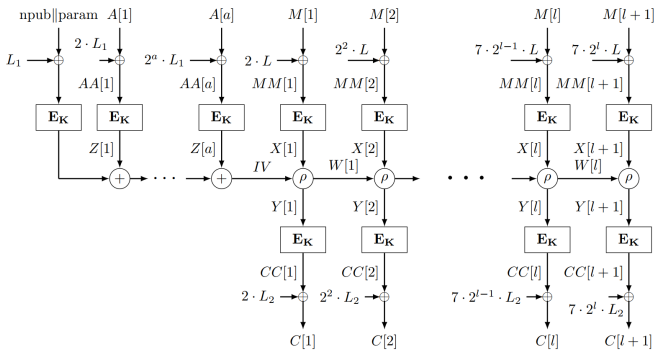

Fig. 4: COLM Encryption mode [3]

identify the key with very high probability.

V. Adapting DFA on COLM

COLM is the second choice of the CAESAR competition for high security applications. It is based on AES-128 as an underlying block cipher. It has two variants COLM_0 and COLM_{127}. Both of them use 64-bit nonce and 128-bit key. They differ in the masking function and the tag size, as COLM_{127} generates several intermediate tags. Our attack targets COLM_0. However, since both vairants share the same structure, the attack should be easily adaptable to both variants. The goal of this section is develop a DFA analysis against COLM, that requires only two encryption calls, and as few fault injections in the faulty encryption call as possible. Moreover, only a single-byte fault is permitted per call. While the DFA against AES is a well studied topic, adapting these attacks to COLM is not straightforward, for two reasons, which can viewed in Figure 4 from [3]:

1) The final output of the AES is XORed with $2^i \cdot L$, where i is the a block counter, and L is a randomized 128-bit parameter derived from the master key K using a secure key Derivation Function (KDF). Hence, the DFA attacks on the last round of AES-128 will return candidates for $K_{10} \oplus 2^i L$, as opposed to K_{10}.
2) Since we can't directly get candidates for the last round key, we cannot use the information from deeper round to further filter out some of the key candidates.

Hence, we need to adapt the AES analysis from [10] to the current scenario. If we look at the results of applying a single-byte random fault at the input of Mix-Column in round 8, we get on average $\approx 2^{32}$ candidates for $K_{10} \oplus 2^i L$, divided as 4 groups of 4 bytes each, each group has $\approx 2^8$ candidates. It was shown in [10] that the complexity of this step is $\approx 2^8$ single round decryption, or 2^{10} 32-bit XORs, if the JDDT of AES is precomputed. If we repeat this for two consecutive blocks, we get 2^{32} equations of the form

$$K_{10} \oplus 2^i \cdot L = A \tag{1}$$

and another set of 2^{32} equations of the form

$$K_{10} \oplus 2^{i+1} \cdot L = B \tag{2}$$

If we try to find every possible candidate for K_{10}, we have to consider $\approx 2^{64}$ linear systems, and we end up with 2^{64} candidates, which is not practical. Hence, we use the fact that the candidates from each block are divided into 4 independent sets and employ a divide and conquer approach to statistically identify the key. For example, we show below the fault propagation in the third the columns in round 9, and how it affects the last round key.

$$\begin{bmatrix} - & - & \delta & - \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \end{bmatrix} \xrightarrow{MC+ARK} \begin{bmatrix} - & - & \delta_1 & - \\ - & - & \delta_2 & - \\ - & - & \delta_3 & - \\ - & - & \delta_4 & - \end{bmatrix}$$

$$\xrightarrow{SB} \begin{bmatrix} - & - & \Delta_1 & - \\ - & - & \Delta_2 & - \\ - & - & \Delta_3 & - \\ - & - & \Delta_4 & - \end{bmatrix} \xrightarrow{SR} \begin{bmatrix} - & - & \Delta_1 & - \\ - & \Delta_2 & - & - \\ \Delta_3 & - & - & - \\ - & - & - & \Delta_4 \end{bmatrix}$$

We can see that the four bytes in this group are XORed with bytes 15, 2, 5 and 8 of the last round key. Hence, the first block give equations of the form:

$$K_{10}[127:120]||K_{10}[23:16]||K_{10}[47:40]||K_{10}[71:64] \oplus X[127:120]||X[23:16]||X[47:40]||X[71:64] = A[31:0] \tag{3}$$

where $X = 2^i \cdot L$, while the next block gives an equation of the form

$$K_{10}[127:120]||K_{10}[23:16]||K_{10}[47:40]||K_{10}[71:64] \oplus X[126:119]||X[22:15]||X[46:39]||X[70:63] = B[31:0] \tag{4}$$

Unlike the original overall system, each potential system of equations considered by Equations 3 and 4 represents an under-defined system of equations with 32 equations and 36 variables. Moreover, the number of potential systems is $(2^8)^2 = 2^{16}$. In conclusion, we can solve every possible system, generating 16 candidates for K_{10} per system and 2^{20} candidates in total. These candidates follow the following form:

$$\begin{aligned} &K_{10}[127] = A[31] \oplus x \\ &K_{10}[23] = A[23] \oplus y \\ &K_{10}[47] = A[15] \oplus z \\ &K_{10}[71] = A[7] \oplus l \\ &\forall 126 \geqslant i \geqslant 120, K_{10}[i] = K_{10}[i+1] \oplus B[i-95] \oplus A[i-96] \\ &\forall 22 \geqslant i \geqslant 20, K_{10}[i] = K_{10}[i+1] \oplus B[i+1] \oplus A[i] \\ &\forall 46 \geqslant i \geqslant 40, K_{10}[i] = K_{10}[i+1] \oplus B[i-31] \oplus A[i-32] \\ &\forall 70 \geqslant i \geqslant 64, K_{10}[i] = K_{10}[i+1] \oplus B[i-63] \oplus A[i-64] \end{aligned} \tag{5}$$

Where x, y, z, l are 4 free variables. Since, this leads to 2^{80} candidates for the full key, it is not enough to make identifying the key practical, so we consider three faults instead of two, injecting a similar fault in the third consecutive block. We notice that the relation between the candidates from the

TABLE I

Target	No. of Faults	Key Complexity
AES-128	1/2	2^8-$2^{32}/2^0$
AES-OCB-128	1/2	2^8-$2^{32}/2^0$
Deoxys	1/2	2^8-$2^{32}/2^0$
AES-COLM-128	1/2/3	$2^{128}/2^{80}/2^0$

second and third blocks (Equation 6, is similar to the relation between candidates from the first and second blocks. Under the assumption that there is only one correct solution and the all the wrong solutions are uniformally distributed, we expect the intersection between the solutions of Equations 3 and 4 and the solutions of Equations 4 and 6 to be $2^{40}/2^{32} = 2^8$, leading to 2^{32} key candidates in total.

$$K_{10}[127:120]||K_{10}[23:16]||K_{10}[47:40]||K_{10}[71:64]\oplus X[125:118]||X[21:14]||X[45:38]||X[69:62] = C[31:0] \quad (6)$$

Moreover, we can further filter out the key candidates by considering the set of solutions of Equations 3 and 6. We note that in this case, we have 40 variables and 32 equations. Hence, we get 256 solutions per system and 2^{24} solutions in total. The intersection between this set of solutions and the previous two sets is expected to be $(2^8 * 2^{24})/2^{32} = 1$, which means that each group will have either 1 candidate (the correct one) or 2 candidates (one correct candidate and one wrong candidate) on average. Hence, the number of master key candidates is expected to be between 1 and 16, which is small enough. We can the brute force over all the final candidates without fault injection (in case more than one candidate is left).

We have verified our analysis by simulating the attack on COLM in software, finding candidates for A, B and C using the JDDT of AES and solving the resulting equations. Our results show that the average number of solutions at every step follows our estimations. Moreover, we were able to identify the master key uniquely using only 3 faults. A notable observation is that, although the size of final intersection set generated by our software was greater than 1, some times all the entries in this set were the same (some of the wrong solutions were equal to the correct solution, due to the selection of the free variables). The overall complexity of the analysis is dominated by the step of finding the intersection between the sets of solutions, which is $\mathcal{O}(nlog\ n)$, with $n \approx 2^{20}$, thanks to the speed-up of the AES analysis using the JDDT.

VI. Conclusions

In this paper, we study DFA attacks on some of the CAESAR competition winners. We study the challenges imposed by the design of these modes, such as masking of the ciphertext. We also show that a very small number of nonce repetition and faults is required, which makes it very practical. The extended attack is summarised in the Table I.

References

[1] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness. https://competitions.cr.yp.to/caesar.html, accessed: 2019-04-10

[2] Lightweight Cryptography. https://csrc.nist.gov/projects/lightweight-cryptography, accessed: 2019-04-10

[3] Andreeva, E., Bogdanov, A., Datta, N., Luykx, A., Mennink, B., Nandi, M., Tischhauser, E., Yasuda, K.: Colm v1 (2016). Submission to the CAESAR competition

[4] Ashur, T., Dunkelman, O., Luykx, A.: Boosting authenticated encryption robustness with minimal modifications. In: Annual International Cryptology Conference. pp. 3–33. Springer (2017)

[5] Berti, F., Guo, C., Pereira, O., Peters, T., Standaert, F.X.: TEDT, for High (Physical) Security Applications

[6] Biham, E., Shamir, A.: Differential fault analysis of secret key cryptosystems. In: Annual international cryptology conference. pp. 513–525. Springer (1997)

[7] Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1. 2. Submission to the CAESAR Competition (2016)

[8] Jean, J., Nikolić, I., Peyrin, T.: Tweaks and keys for block ciphers: the tweakey framework. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 274–288. Springer (2014)

[9] Jean, J., Nikolic, I., Peyrin, T., Seurin, Y.: Deoxys v1. 41. Submitted to CAESAR (2016)

[10] Khairallah, Mustafa and Hou, Xiaolu and Najm, Zakaria and Breier, Jakub and Bhasin, Shivam and Peyrin, Thomas: SoK: On the DFA Vulnerabilities of SPNs. In: AsiaCCS 2019 (2019)

[11] Krovetz, T., Rogaway, P.: Ocb (v1.1). Submission to the CAESAR competition: https://competitions. cr. yp. to/round3/ocbv11. pdf (2016)

[12] Wu, H.: Acorn: a lightweight authenticated cipher (v3). Candidate for the CAESAR Competition. See also https://competitions. cr. yp. to/round3/acornv3. pdf (2016)

[13] Wu, H., Preneel, B.: Aegis: a fast authenticated encryption algorithm. In: International Conference on Selected Areas in Cryptography. pp. 185–201. Springer (2013)