The Deoxys AEAD Family

Jérémy Jean[1], Ivica Nikolić[2], Thomas Peyrin[3], and Yannick Seurin[1]

[1] ANSSI, Paris, France
{Jeremy.Jean,Yannick.Seurin}@ssi.gouv.fr

[2] National University of Singapore, Singapore
cube444@gmail.com

[3] Nanyang Technological University, Singapore
Thomas.Peyrin@ntu.edu.sg

https://sites.google.com/view/deoxyscipher

Abstract. We present the Deoxys family of authenticated encryption schemes, which consists of Deoxys-I and Deoxys-II. Both are nonce-based authenticated encryption schemes with associated data and have either 128- or 256-bit keys. Deoxys-I is similar to OCB: it is single-pass but insecure when nonces are repeated; in contrast, Deoxys-II is nonce-misuse resistant. Deoxys-II was selected as first choice in the final portfolio of the CAESAR competition for the *defense-in-depth* category.
Deoxys uses a new family of tweakable block ciphers as internal primitive, Deoxys-TBC, which follows the TWEAKEY framework (Jean, Nikolić, and Peyrin, ASIACRYPT 2014) and relies on the AES round function.
Our benchmarks indicate that Deoxys does not sacrifice efficiency for security and performs very well both in software (e.g., Deoxys-I efficiency is similar to AES-GCM) and hardware.

1 Introduction

Authenticated Encryption. Confidentiality and authenticity are the two main security properties one wants to achieve when protecting data. These two security goals have long been studied independently, giving rise to encryption schemes on one hand and MAC (authentication) schemes on the other hand. While it is known how to securely combine these two primitives [7, 52], in a large number of practical cases this has led to catastrophic failures [1, 2, 43, 63]. For this reason, the cryptographic community has striven to provide so-called *authenticated encryption* schemes which simultaneously ensure confidentiality and authenticity and are therefore less prone to misuse.

In order to foster the research efforts in this domain, the CAESAR competition[4] was engaged in 2014. After four years of public scrutiny, the CAESAR committee selected final portfolios of authenticated encryption schemes for three use cases:

[4] https://competitions.cr.yp.to/caesar-submissions.html

- lightweight applications (resource constrained environments),
- high-performance applications,
- defense in depth.

THE Deoxys FAMILY. In this work, we propose Deoxys, a family of nonce-based authenticated encryption schemes with associated data. It consists of Deoxys-I, which is secure only when nonces are not repeated, and Deoxys-II which retains security even when nonces are repeated (more precisely, it achieves the MRAE security notion of Rogaway and Shrimpton [60]). Both can be used either with 128- or 256-bit keys. Deoxys-II was selected as main choice for the defense-in-depth portfolio of the CAESAR competition.

Deoxys is based on a new family of tweakable block ciphers, Deoxys-TBC, which uses the well-studied AES round function as a building block. Unlike tweakable block ciphers built from conventional block ciphers using a mode of operation (such as XE/XEX [58]), Deoxys-TBC is an ad-hoc design following the so-called TWEAKEY framework [36] which unifies the treatment of the key and the tweak inputs of a tweakable block cipher.

We use this TBC in two different yet similar, fully parallel, and provably secure authenticated encryption modes: for Deoxys-I, we use a mode which is very similar to TAE [47] and ΘCB3 [44]; for Deoxys-II, we adopt SCT-2, a variant of the SCT mode proposed in [55] and inspired by SIV [60].

SECURITY. Deoxys-TBC offers a very good security margin. Deoxys-TBC-256 has 14 rounds and Deoxys-TBC-384 has 16 rounds. Being AES-based, Deoxys benefits from the vast literature on the cryptanalysis of the AES. The best known attacks on AES-based designs in the secret-key security model for similar size of keys reach 7 to 9 rounds. In the related-key security model, AES-192 and AES-256 are theoretically broken [11, 12]. With 14 rounds or more, the two versions of Deoxys-TBC offer a comfortable security margin regarding this class of attacks. Interestingly, Deoxys-TBC-256 is very similar to AES-256 for a fixed tweak value, but the improved key schedule significantly reduces the number of rounds required for security: Deoxys-TBC-256 only needs 12 rounds to be secure against related-key attacks, whereas AES-256 (14 rounds) is already vulnerable to a theoretical related-key distinguisher. Deoxys-TBC has been been analyzed by several third parties during the CAESAR competition, which reinforces the confidence one can have in its security. We detail this in the section dedicated to the security analysis.

The modes of operations used in Deoxys are provably secure. For Deoxys-I, we rely on the existing security proof for ΘCB3 [44]. The mode has "perfect" security, meaning that when used with a uniformly random (tweakable) permutation, the attacker's advantage is zero for breaking confidentiality and $2^{-\tau}$ for breaking authenticity, where τ is the tag length. Moreover, Deoxys-TBC being an ad-hoc design, it is not subject to birthday-bound attacks that affect most generic constructions such as XE/XEX [58].[5] For Deoxys-II, we provide a detailed security

[5] Albeit at the cost of a stronger security assumption on the primitive.

proof of the SCT-2 mode in this paper. A notable feature of this mode is that it provides beyond-birthday-bound security when nonces are only moderately repeated and graceful degradation of the security bounds with the maximal number of nonce repetitions.

Being AES-based, existing techniques for protecting AES against side-channel attacks can be straightforwardly adapted to Deoxys-TBC.

Performances. Deoxys achieves very good performances in software. The number of calls to the internal primitive is close to optimal (one call per block for Deoxys-I and two calls per block for Deoxys-II, plus one extra-call for tag computation). Being AES-based, it greatly benefits from the AES-NI instruction set added in the latest processors. In addition, as we use fully parallelizable modes, the cycles per byte count drops significantly. For example, Deoxys-I has performances close to AES-GCM [48], although AES-GCM ensures only birthday-bound security.

Deoxys does not require any pre-computation and hence is very efficient for small messages, which is particularly important in many lightweight applications where messages are usually a few dozens of bytes long or for Internet traffic as packets sizes can be rather small. In contrast, sponge-based or stream cipher-based designs like Ascon [25], ACORN [68], AEGIS [69], FIDES [10] (broken in [24]) or ALE [16] (broken in [41]) usually require a costly initialization.

In hardware, Deoxys can be implemented with a limited area overhead using existing AES lightweight implementations, the extra area mainly consisting in 192 extra bits of memory for the mode and to store the tweak. The key can be hardcoded for smaller area footprint.

Further Variants. In addition to the four schemes Deoxys-I/II-128/256 originally submitted to the CAESAR competition, we propose here two new 128-bit key ones called Deoxys-AE1 and Deoxys-AE2, which are more efficient variants of respectively Deoxys-I-128 and Deoxys-II-128. These variants are furthermore conceptually simpler and they provide more flexible parameter sets. The idea is to use Deoxys-TBC-384 instead of Deoxys-TBC-256, while keeping the key size at 128 bits: this provides more tweak space which can be used to process more associated data and message bits per TBC call, allow a larger nonce and counter in Deoxys-I-128, fully separate the counter from the nonce in Deoxys-II-128 (for better independence).

Organization of the Paper. We start with introducing the general notation and standard security notions in Section 2. In Section 3, we provide the specification of Deoxys, starting with Deoxys-TBC and then the operating modes for Deoxys-I and Deoxys-II. In Section 4, we detail the security claims for various scenarios and parameters. In Section 5, we explain some design decisions regarding Deoxys-TBC and in Section 6 we perform some security analysis regarding this new TBC. In Section 7, we give the security proof for SCT-2, the mode of operation used in Deoxys-II. Finally, we provide software and hardware implementation performances in respectively Section 8 and Section 9.

2 Preliminaries

2.1 General Notation

We let $\{0,1\}^*$ denote the set of all bit strings and $\{0,1\}^{\leq \ell}$ denote the set of all bit strings of length at most ℓ. The empty string is denoted ϵ. The length of a bit string X is denoted $|X|$. The concatenation of two bit strings X and Y is denoted $X\|Y$. If X and Y are respectively n-bit and m-bit strings, $n < m$, then $X \oplus Y$ denotes the n-bit string obtained by xoring X with the n leftmost bits of Y. Given some parameter n, we let ozpad_n (or ozpad when the parameter n is implicit) denote the padding function defined for $X \neq \epsilon$ as

$$\mathsf{ozpad}(X) \coloneqq \begin{cases} X & \text{if } |X| = n \\ X\|1\|0^{n-|X|-1} & \text{if } |X| < n. \end{cases}$$

Given a bit string X of length i or larger, the i leftmost bits of X are denoted $\lceil X \rceil_i$ and the i rightmost bits of X are denoted $\lfloor X \rfloor_i$. We let $X \lll a$ denote the bit string X rotated by a positions to the left.

2.2 Tweakable Block Ciphers

A *tweakable block cipher* (TBC) with key space $\mathcal{K}$, tweak space $\mathcal{T}$, and domain $\mathcal{X}$ is a mapping $E : \mathcal{K} \times \mathcal{T} \times \mathcal{X} \to \mathcal{X}$ such that for any key $K \in \mathcal{K}$ and any tweak $T \in \mathcal{T}$, $E(K, T, \cdot)$ is a permutation of $\mathcal{X}$. We often write $E_K(T, X)$ or $E_K^T(X)$ in place of $E(K, T, X)$. Given $K \in \mathcal{K}$ and $T \in \mathcal{T}$, we let $E_K^{-1}(T, \cdot)$ denote the inverse of $X \mapsto E_K(T, X)$. We let $\mathsf{TBC}(\mathcal{K}, \mathcal{T}, \mathcal{X})$ denote the set of all tweakable block ciphers with key space $\mathcal{K}$, tweak space $\mathcal{T}$, and domain $\mathcal{X}$. A *tweakable permutation* with tweak space $\mathcal{T}$ and domain $\mathcal{X}$ is a mapping $\widetilde{P} : \mathcal{T} \times \mathcal{X} \to \mathcal{X}$ such that for any tweak $T \in \mathcal{T}$, $X \mapsto \widetilde{P}(T, X)$ is a permutation of $\mathcal{X}$. We often write $\widetilde{P}^T(X)$ in place of $\widetilde{P}(T, X)$. We let $\mathsf{TP}(\mathcal{T}, \mathcal{X})$ denote the set of all tweakable permutations with tweak space $\mathcal{T}$ and domain $\mathcal{X}$. The security of a TBC is defined as follows.

Definition 1 ((S)TPRP security). *Let $E \in \mathsf{TBC}(\mathcal{K}, \mathcal{T}, \mathcal{X})$ and A be an adversary with oracle access to a tweakable permutation with tweak space $\mathcal{T}$ and domain $\mathcal{X}$ and possibly its inverse. The advantage of A in breaking the TPRP-security of E is defined as*

$$\mathbf{Adv}_E^{\mathrm{TPRP}}(\mathsf{A}) = \left| \Pr\left[K \leftarrow_{\$} \mathcal{K} : \mathsf{A}^{E_K} = 1\right] - \Pr\left[\widetilde{P} \leftarrow_{\$} \mathsf{TP}(\mathcal{T}, \mathcal{X}) : \mathsf{A}^{\widetilde{P}} = 1\right] \right|.$$

The advantage of A in breaking the strong TPRP-security (STPRP-security) of E is defined as

$$\mathbf{Adv}_E^{\mathrm{STPRP}}(\mathsf{A}) = \Big| \Pr\left[K \leftarrow_{\$} \mathcal{K} : \mathsf{A}^{E_K, E_K^{-1}} = 1\right] - \Pr\left[\widetilde{P} \leftarrow_{\$} \mathsf{TP}(\mathcal{T}, \mathcal{X}) : \mathsf{A}^{\widetilde{P}, \widetilde{P}^{-1}} = 1\right] \Big|.$$

Let E be a TBC with tweak space of the form $\mathcal{T}' = \mathcal{I} \times \mathcal{T}$ for some subset $\mathcal{I} \subset \mathbb{N}$ and some set $\mathcal{T}$. We call $\mathcal{T}$ the *effective* tweak space of E. Then, for $i \in \mathcal{I}$, we let E^i denote the tweakable block cipher with the same key and message spaces as E and tweak space $\mathcal{T}$ defined by

$$E^i(K, T, X) = E(K, (i, T), X).$$

By the same convention as before, we sometimes write $E^i_K(T, X)$ or $E^{i,T}_K(X)$ for $E^i(K, T, X)$. Clearly, when E is an ideal TBC drawn uniformly at random from $\mathsf{TBC}(\mathcal{K}, \mathcal{T}', \mathcal{M})$, then each E^i is an independent ideal TBC drawn uniformly at random from $\mathsf{TBC}(\mathcal{K}, \mathcal{T}, \mathcal{M})$.

In practice, in all our modes, E has tweak space $\mathcal{T}' = \{0,1\}^t$ and we use 4-bit prefixes (interpreted as integers) for tweak separation, so that the effective tweak space is $\{0,1\}^{t-4}$.

2.3 Authenticated Encryption

A nonce-based authenticated encryption scheme with associated data (*nAE scheme* for short) is a tuple $\Pi = (\mathcal{K}, \mathcal{N}, \mathcal{A}, \mathcal{M}, \mathcal{C}, \mathsf{Enc}, \mathsf{Dec})$, where $\mathcal{K}$, $\mathcal{N}$, $\mathcal{A}$, $\mathcal{M}$, and $\mathcal{C}$ are non-empty sets of bit strings with $\mathcal{K}$ and $\mathcal{N}$ finite and Enc and Dec are deterministic algorithms. The encryption algorithm Enc takes as input a key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$ (also called *public message number* in CAESAR terminology), associated data $A \in \mathcal{A}$, and a message $M \in \mathcal{M}$, and outputs a ciphertext $C \in \mathcal{C}$ and a bit string $\mathsf{tag} \in \{0,1\}^\tau$ for some integer $\tau \geq 0$ (we assume that Enc returns $\perp$ if one of the inputs is not in the intended set). The decryption algorithm Dec takes as input a key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, associated data $A \in \mathcal{A}$, a ciphertext $C \in \mathcal{C}$, and a string $\mathsf{tag} \in \{0,1\}^\tau$, and outputs either a message $M \in \mathcal{M}$, or a special symbol $\perp$ indicating that decryption failed. We require that $\mathsf{Dec}(K, N, A, \mathsf{Enc}(K, N, A, M)) = M$ for all tuples $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$. We also require that if $\mathcal{M}$ contains a bit string of length m, then it contains all bit strings of length m, and that $|\mathsf{Enc}(K, N, A, M)| = |M|$ for all $(K, N, A, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M}$. We write $\mathsf{Enc}_K(N, A, M)$ for $\mathsf{Enc}(K, N, A, M)$ and $\mathsf{Dec}_K(N, A, C, \mathsf{tag})$ for $\mathsf{Dec}(K, N, A, C, \mathsf{tag})$.

Definition 2 (nAE-security). *Let $\Pi = (\mathcal{K}, \mathcal{N}, \mathcal{A}, \mathcal{M}, \mathcal{C}, \mathsf{Enc}, \mathsf{Dec})$ be a nAE scheme. The advantage of an adversary A in breaking Π is defined as*

$$\mathbf{Adv}^{\mathrm{nAE}}_{\Pi}(\mathsf{A}) = \left|\Pr\left[K \leftarrow_{\$} \mathcal{K} : \mathsf{A}^{\Pi.\mathsf{Enc}_K,\, \Pi.\mathsf{Dec}_K} = 1\right] - \Pr\left[\mathsf{A}^{\mathsf{Rand},\, \mathsf{Rej}} = 1\right]\right|,$$

where Rand is an oracle which on input $(N, A, M) \in \mathcal{N} \times \mathcal{A} \times \mathcal{M}$ outputs a uniformly random[6] element in $\{0,1\}^{|M|} \times \{0,1\}^\tau$ and Rej is an oracle which always outputs $\perp$. The adversary is not allowed to make a decryption query (N, A, C, tag) if a previous encryption query (N, A, M) returned (C, tag).

The adversary is said nonce-respecting if it never repeats a nonce $N \in \mathcal{N}$ in its queries to the encryption oracle.

[6] We assume that Rand returns the same output if a query is repeated.

3 Specification

In this section, we present the full specification of the Deoxys family of nAE schemes. In total, we specify six schemes:

- Deoxys-I-128 and Deoxys-I-256, which are suitable only in the nonce-respecting setting, i.e., users must never repeat a nonce in two different encryption calls with the same key;
- Deoxys-II-128 and Deoxys-II-256, which are nonce-misuse resistant, i.e., security is retained even when nonces are repeated;
- Deoxys-AE1, a more efficient, flexible and cleaner variant of Deoxys-I-128;
- Deoxys-AE2, a more efficient, flexible and cleaner variant of Deoxys-II-128.

Suffix I indicates that any nonce must be used at most once, while suffix II indicates that even if nonces repeat, a well-defined level of security is retained.

Each member of the family is designed by combining one tweakable block cipher(s) and a mode of operation. First, we specify the family of tweakable block ciphers Deoxys-TBC on which Deoxys is built in Section 3.2. Then, we specify the modes of operations for the Deoxys-I subfamily in Section 3.3 and for the Deoxys-II subfamily in Section 3.4. The two modes are quite similar, the main difference being that the first one applies one pass on the message blocks for both encryption and decryption, while the second one performs two passes for encryption (which is necessary to obtain nonce-misuse resistance [60]) and one pass for decryption.[7]

3.1 Parameters

In all the following, we use the following notation. The internal tweakable block cipher E has key space $\mathcal{K} = \{0,1\}^k$, tweak space $\mathcal{T} = \{0,1\}^t$, and domain $\mathcal{X} = \{0,1\}^n$. The nAE scheme based on E has the same key space $\mathcal{K} = \{0,1\}^k$, nonce space $\{0,1\}^{|N|}$, AD, message, and ciphertext spaces $\mathcal{A} = \mathcal{M} = \mathcal{C} = \{0,1\}^{\leq n \cdot \max_\ell}$, and tag length $\tau \in [0,n]$.[8]

The concrete parameters for Deoxys-TBC are given in Table 1, while parameters for the Deoxys nAE schemes are given in Table 2.

3.2 The Tweakable Block Cipher Family Deoxys-TBC

Deoxys-TBC is a family of tweakable block ciphers with two members:

- Deoxys-TBC-256 for which the sum of the sizes of the key and the tweak is 256 bits and which is used for Deoxys-I-128 and Deoxys-II-128

[7] Despite being one-pass, decryption makes two primitive calls per message blocks as in SIV [60]. Misuse resistant schemes with rate-1 decryption have been recently proposed [49].

[8] Tags can be truncated to $\tau < n$ bits but we recommend using $\tau = n$.

Table 1: Parameters for the Deoxys-TBC family of tweakable block ciphers: k is the key size, t is the tweak size, and n is the block size.

Name	$k+t$	n	#rounds
Deoxys-TBC-256	256	128	14
Deoxys-TBC-384	384	128	16

Table 2: Parameters for the Deoxys family of nAE schemes: k is the key size, $|N|$ is the nonce size, $\max_\ell$ is the maximal message size (in blocks of 128 bits), τ is the recommended tag size, and the last column indicates which variant of Deoxys-TBC is used.

Name	k	$\|N\|$	$\max_\ell$	τ	Deoxys-TBC
Deoxys-I-128	128	64	2^{60}	128	256
Deoxys-I-256	256	64	2^{60}	128	384
Deoxys-AE1	128	128	2^{120}	128	384
Deoxys-II-128	128	120	2^{60}	128	256
Deoxys-II-256	256	120	2^{60}	128	384
Deoxys-AE2	128	128	2^{120}	128	384

- Deoxys-TBC-384 for which the sum of the sizes of the key and the tweak is 384 bits and which is used for Deoxys-I-256, Deoxys-II-256, Deoxys-AE1 and Deoxys-AE2.

Deoxys-TBC is an AES-like design, i.e., it is a substitution-permutation network (SPN) that transforms the plaintext into the ciphertext through a sequence of round functions that depend on the key and the tweak. As most AES-like designs, the state of Deoxys-TBC is seen as 4×4 matrix of bytes (we let c denote the size of a cell in bits, i.e. $c = 8$). We let $\mathbb{K}$ denote the finite field $GF(2^8)$ defined by the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$.

The number r of rounds is 14 for Deoxys-TBC-256 and 16 for Deoxys-TBC-384. One round, similarly to a round in AES, consists of the following four transformations applied to the internal state in the order specified below:

- AddRoundTweakey: XOR the 128-bit round subtweakey (defined further) to the internal state,
- SubBytes: Apply the 8-bit S-box $\mathcal{S}$ of AES to the 16 bytes of the internal state (see definition in Appendix A.1),
- ShiftRows: Rotate the 4-byte i-th row left by $\rho[i]$ positions, where $\rho = (0, 1, 2, 3)$.
- MixBytes: Multiply the internal state by the 4×4 constant MDS matrix $\mathbf{M}$ defined below.

After the last round, a final AddRoundTweakey operation is performed to produce the ciphertext.

The MDS matrix $\mathbf{M}$ we use is the one from the AES (coefficients are in $\mathbb{K}$):

$$\mathbf{M} = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}.$$

The round function f^{-1} for decryption applies the inverse of the four transformations in the reverse order (with subtweakeys in reverse order as well). Namely, we perform r times the following operations:

- InvAddRoundTweakey: XOR the 128-bit round subtweakey to the internal state,
- invMixBytes: Multiply the internal state by the 4×4 MDS matrix $\mathbf{M}^{-1}$,
- InvShiftRows: Rotate the 4-byte i-th row *right* by $\rho[i]$ positions, where $\rho = (0, 1, 2, 3)$,
- InvSubBytes: Apply the inverse 8-bit S-box S^{-1} to the 16 bytes of the internal state (see definition in Appendix A.1).

Finally, a final InvAddRoundTweakey operation is performed to produced the plaintext value. For the sake of completeness, we provide the inverse of the $\mathbf{M}$ matrix:

$$\mathbf{M}^{-1} = \begin{pmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{pmatrix}.$$

Definition of the Subtweakeys. So far, the description of the cipher has followed the classical construction of an AES-like block cipher. The operation AddRoundTweakey, and in particular the production of the subtweakeys, is where Deoxys-TBC differs from the AES.

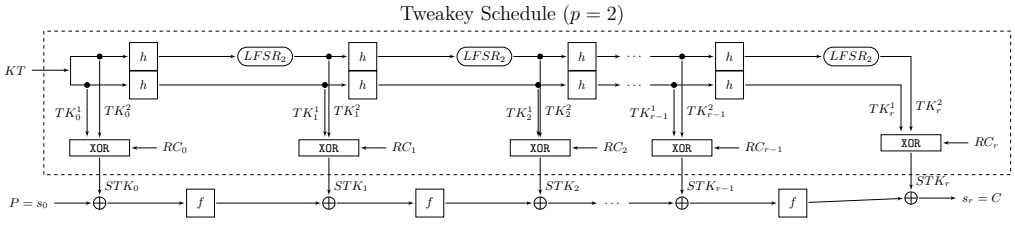


Fig. 1: Instantiation of the TWEAKEY framework for Deoxys-TBC.

We write the concatenation of the key K and the tweak T as KT, i.e., $KT = K \| T$. Then, the tweakey state is divided into words of 128 bits. More precisely,

in Deoxys-TBC-256, the size of KT is 256 bits with the first (most significant) 128 bits of KT being denoted W_2, while the second W_1. For Deoxys-TBC-384, the size of KT is 384 bits, with the first (most significant) 128 bits of KT being denoted W_3, the second W_2 and the third W_1. Finally, we let STK_i denote the subtweakey (a 128-bit word) that is added to the state at Round i of the cipher during the AddRoundTweakey operation. For Deoxys-TBC-256, a subtweakey is defined as:

$$STK_i = TK_i^1 \oplus TK_i^2 \oplus RC_i,$$

whereas for the case of Deoxys-TBC-384 it is defined as:

$$STK_i = TK_i^1 \oplus TK_i^2 \oplus TK_i^3 \oplus RC_i.$$

The 128-bit words TK_i^1, TK_i^2, TK_i^3 are outputs produced by a tweakey schedule algorithm, initialized with $TK_0^1 = W_1$ and $TK_0^2 = W_2$ for Deoxys-TBC-256 and with $TK_0^1 = W_1$, $TK_0^2 = W_2$ and $TK_0^3 = W_3$ for Deoxys-TBC-384. The tweakey schedule algorithm is defined as

$$\begin{aligned} TK_{i+1}^1 &= h(TK_i^1), \\ TK_{i+1}^2 &= h(LFSR_2(TK_i^2)), \\ TK_{i+1}^3 &= h(LFSR_3(TK_i^3)), \end{aligned}$$

where the byte permutation h is defined as:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 1 & 6 & 11 & 12 & 5 & 10 & 15 & 0 & 9 & 14 & 3 & 4 & 13 & 2 & 7 & 8 \end{pmatrix},$$

and where we number the 16 bytes of a 128-bit tweakey word by the usual ordering:

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix}.$$

The transformation can therefore be described as:

$$\begin{pmatrix} 0 & 4 & 8 & 12 \\ 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & 5 & 9 & 13 \\ 6 & 10 & 14 & 2 \\ 11 & 15 & 3 & 7 \\ 12 & 0 & 4 & 8 \end{pmatrix}.$$

The $LFSR_2$ and $LFSR_3$ functions are simply the application of an LFSR to each of the 16 bytes of a tweakey 128-bit word. More precisely, the two LFSRs used are given in Table 3 (x_0 stands for the LSB of the cell).

Table 3: The two LFSRs used in `Deoxys-TBC` tweakey schedule.

$LFSR_2$	$(x_7 \| x_6 \| x_5 \| x_4 \| x_3 \| x_2 \| x_1 \| x_0) \rightarrow (x_6 \| x_5 \| x_4 \| x_3 \| x_2 \| x_1 \| x_0 \| x_7 \oplus x_5)$
$LFSR_3$	$(x_7 \| x_6 \| x_5 \| x_4 \| x_3 \| x_2 \| x_1 \| x_0) \rightarrow (x_0 \oplus x_6 \| x_7 \| x_6 \| x_5 \| x_4 \| x_3 \| x_2 \| x_1)$

Finally, RC_i are the key schedule round constants, and are defined as:

$$RC_i = \begin{pmatrix} 1 & \texttt{RCON}[i] & 0 & 0 \\ 2 & \texttt{RCON}[i] & 0 & 0 \\ 4 & \texttt{RCON}[i] & 0 & 0 \\ 8 & \texttt{RCON}[i] & 0 & 0 \end{pmatrix}$$

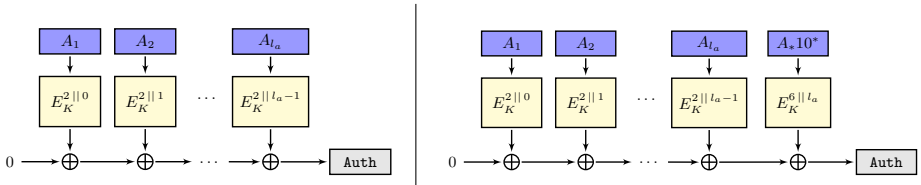
where $\texttt{RCON}[i]$ denotes the $i+15$-th key schedule constant of the `AES` whose actual values are given in Appendix B.

Cipher Instances Separation. We note that the tweak and key material are not made explicitly distinct in `Deoxys-TBC`, and one might argue that since the tweakable block cipher is always the same whatever is the amount of key or tweak inputs, there are some obvious relations between these different cipher variants. Only considering the primitive `Deoxys-TBC`, a simple distinction between the instances could be to encode the parameter sizes into the round constants. We however chose to not consider such related-cipher attacks [67], but instead leave this distinction at the discretion of the protocol or mode calling the primitive. Especially, for the two `Deoxys` operating modes, the separation between the tweakable block cipher instances is naturally and safely done since the tweak and key sizes are fixed and the placement of key and tweak material is fully determined.

3.3 Mode of Operation for `Deoxys-I`

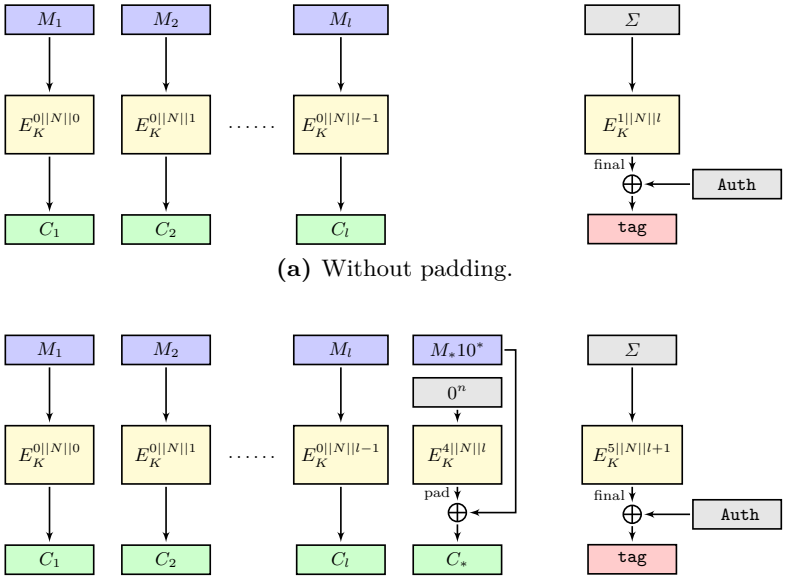
The mode of operation for `Deoxys-I-128` and `Deoxys-I-256` is depicted in Fig. 2, Fig. 3a, and Fig. 3b. A specification in pseudocode is given in appendix in Algorithm 1 (encryption) and Algorithm 2 (decryption). This mode is similar to `TAE` [47] or `ΘCB3` [44] (the tweakable block cipher generalization of `OCB3`) and therefore directly benefits from their security proofs regarding authentication and privacy.

The `Deoxys-AE1` Variant. We remark that the authentication part of the mode could be made more efficient by allowing more data to be processed via the tweak input, similarly to what is done in `ZMAC` [34]. Indeed, instead of using `Deoxys-TBC-256` in `Deoxys-I-128`, one could use instead `Deoxys-TBC-384` during the authentication phase, which directly permits to process 128 extra bits of data via the larger tweak. Besides, by reclaiming the tweak space used by the



(a) Without padding. **(b)** With padding.

Fig. 2: Handling of the associated data for `Deoxys-I-128` and `Deoxys-I-256`: in the case where the associated data is a multiple of the block size, no padding is needed.



(a) Without padding.

(b) With padding.

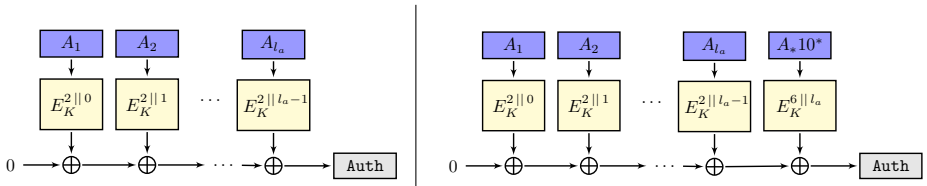
Fig. 3: Message processing for `Deoxys-I-128` and `Deoxys-I-256`: in the case where the message-length is a multiple of the block size, no padding is needed. Note that the checksum Σ is computed with a 10^* padding for block M^*.

constant 0^{64}, we can allow a larger counter size so that very large associated data inputs can be handled. If `Deoxys-TBC-384` is also used in the encryption phase, then a larger nonce as well as counter size can be used to also handle very large message inputs. We call this slightly more efficient version `Deoxys-AE1`. For more flexibility, the domain separation constant has been increased from 4 bits to 8 bits. For completeness, we give an algorithmic description in Algorithm 3 and Algorithm 4 in appendix (and depicted in Figures 10 and 11).

3.4 Mode of Operation for Deoxys-II

The encryption algorithm for Deoxys-II-128 and Deoxys-II-256 is depicted in Fig. 4 and Fig. 5 for the authentication part and in Fig. 6 for the encryption part. A specification in pseudocode is given in appendix in Algorithm 5 (encryption) and Algorithm 6 (decryption). This mode is a variant of SCT (Synthetic Counter in Tweak) proposed in [55] that we call SCT-2. The encryption part is kept unchanged compared with SCT, only the computation of the tag is modified in order to provide graceful degradation of security for authentication with the maximal number of repetitions of nonces [19] (a property that was ensured for confidentiality by SCT, but not for authenticity).

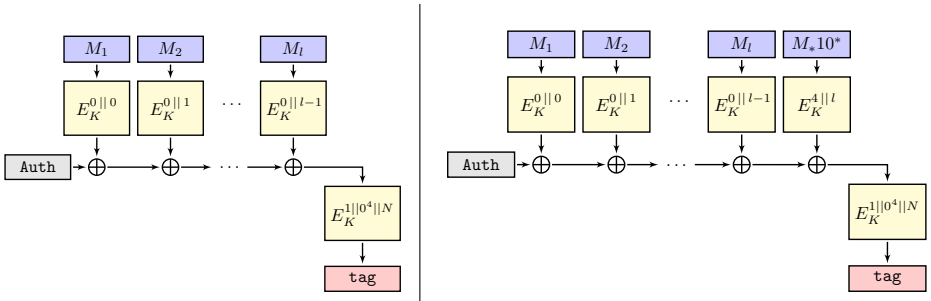
In this SCT-2 variant described below, the nonce N has a size of 120 bits, to include it in the tweak input of the block cipher call producing the tag. If necessary, nonce sizes up to 128 bits can be accommodated at the expense of an additional block cipher call. For this, one simply replaces the finalization of the tag (Lines 21 and 29 in the following algorithms) $\text{tag}' \leftarrow E_K(0001\|0^4\|N, \text{tag}')$ with $\text{tag}' \leftarrow E_K(0001\|0^4\|N', \text{tag}')$, where N' are the, say, 120 leftmost bits of the encryption of N with a reserved 4-bit tweak prefix that is used nowhere else in the mode.



(a) Without padding. **(b)** With padding.

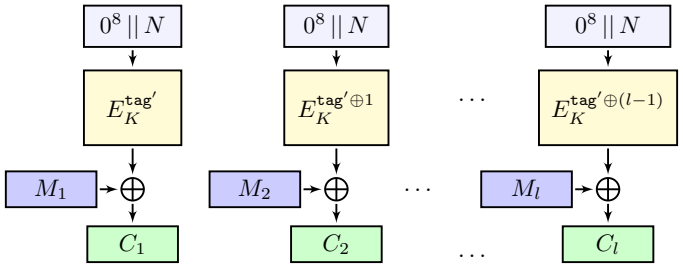
Fig. 4: Handling of the associated data for Deoxys-II-128 and Deoxys-II-256: in the case where the associated data is a multiple of the block size, no padding is needed.

The Deoxys-AE2 Variant. We remark that the authentication part of the mode could be made more efficient by allowing more data to be processed via the tweak input, similarly to what is done in ZMAC [34]. Indeed, instead of using Deoxys-TBC-256 in Deoxys-II-128, one could use instead Deoxys-TBC-384 during the authentication phase, which directly permits to process 128 extra bits of data via the larger tweak. Besides, by reclaiming the tweak space used by the constant 0^{64}, we can allow a larger counter size so that very large inputs can be handled. In the encryption part, using Deoxys-TBC-384 allows to separate completely the tag from the counter, which will improve the security bounds. For completeness, we give an algorithmic description in Algorithm 7 and Algorithm 8 in appendix (and depicted in Figures 12, 13 and 14).

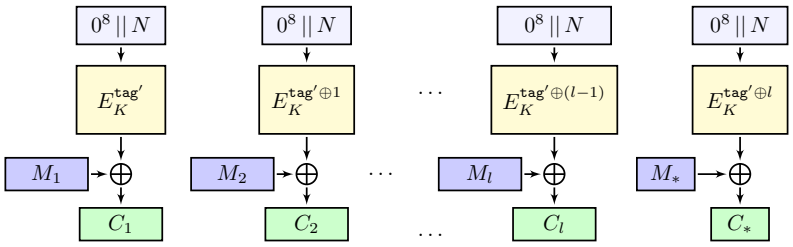


(a) Without padding. **(b)** With padding.

Fig. 5: Message processing in the authentication part of `Deoxys-II-128` and `Deoxys-II-256`: in the case where the message-length is a multiple of the block size, no padding is needed.



(a) Message-length is a multiple of the block size.



(b) Message-length is not a multiple of the block size.

Fig. 6: Message processing for the encryption part of `Deoxys-II-128` and `Deoxys-II-256`. We let $\mathtt{tag}' = \mathtt{tag} \vee 1\|0^{127}$, i.e., the MSB of `tag` is forced to 1. Hence, the 4-bit prefix of the tweak input takes values in $\{8, \ldots, 15\}$.

4 Security Claims

4.1 Claims

We provide our security claims for the different variants of `Deoxys` in Table 4 with respect to the key size k and the block size n.

Table 4: Security claims for `Deoxys`. The upper table stands for the situation where the user will never repeat the same nonce value N for the same key. The lower table stands for the situation where such repetitions in N for the same key are allowed. The bit security of our designs is expressed in terms of calls to the internal tweakable block cipher, up to a small logarithmic factor. Claims for `Deoxys-AE1` and `Deoxys-AE2` are the same as `Deoxys-I` and `Deoxys-II` respectively.

	Security (bits)	
Goal (nonce-respecting setting)	`Deoxys-I`	`Deoxys-II`
Key recovery	k	k
Confidentiality for the plaintext	n	$n-1$
Integrity for the plaintext	n	$n-1$
Integrity for the associated data	n	$n-1$
Integrity for the public message number	n	$n-1$

	Security (bits)	
Goal (nonce-misuse setting)	`Deoxys-I`	`Deoxys-II`
Key recovery	k	k
Confidentiality for the plaintext	none	$n/2$
Integrity for the plaintext	none	$n/2$
Integrity for the associated data	none	$n/2$
Integrity for the public message number	none	$n/2$

We claim full n-bit security for both `Deoxys-I` and `Deoxys-II` in the nonce-respecting setting, in contrast to other modes such as `AES-GCM` [48] or `OCB3` [44], which only ensure birthday-bound security. In the nonce-misuse scenario, we claim a birthday-bound security concerning `Deoxys-II`.

We assume that the maximum number of AD/message pairs that are handled with a given key is $2^{\max_m}$ with $\max_m = 64$ for all variants of `Deoxys`. Moreover, for all variants of `Deoxys-I`, we assume that the total size of the associated data and the message do not exceed $16 \cdot 2^{\max_\ell} = 2^{64}$ bytes. (For `Deoxys-II`, the total size of the associated data and messages cannot exceed $16 \cdot 2^{\max_\ell} \cdot 2^{\max_m} = 2^{128}$ bytes.)

We recommend to use a tag size $\tau = n$. However, in case a smaller tag size is required, the security claims will drop according to τ. We explicitly exclude related-cipher attacks, for example when an attacker would try to find some correlations between different versions of `Deoxys` (we assume that such a separation, if needed, will be handled by the protocol using the authenticated encryption primitive).

4.2 Comparison of Deoxys Modes With Other Modes

Deoxys-I provides "full" security in the nonce-respecting setting; more precisely, confidentiality is perfectly guaranteed and the forgery probability is $2^{-\tau}$, independently of the number of blocks of data in encryption/decryption queries made by the adversary. In comparison, OCB only provides security up to the birthday bound, more precisely up to roughly $2^{n/2}$ *blocks of data* since it relies on XE/XEX (a construction of a tweakable block cipher from a standard block cipher with security only up to the birthday bound). In Fig. 7, we represent the security of several nonce-respecting schemes selected for the third round of the CAESAR competition. To give an numerical example, with 2^{32} messages of 64 KB each (corresponding to the case $\sigma = 2^{44}$ blocks of 16 bytes on the plot), existing security proofs ensure that the attacker against authenticity has an advantage of at most 2^{-37} for OCB3, 2^{-41} for AES-GCM, 2^{-73} for AES-GCM-SIV. For the same amount of data, the advantage becomes 2^{-94} for Deoxys-II, and remains 2^{-128} for Deoxys-I.

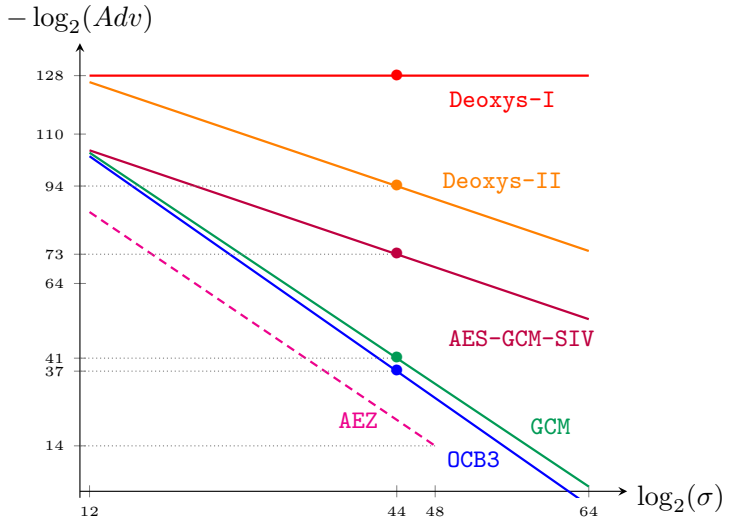


Fig. 7: Security comparison of some nonce-based modes in the nonce-respecting setting. The figure plots the nAE-advantage of an adversary as a function of the total number of queried blocks σ when the message length is fixed to $\ell = 2^{12}$ blocks.

Deoxys-II provides "full" security in the nonce-misuse setting, in the MRAE sense of [60] (but its bound depends on the number of queries made by the adversary). Moreover, security degrades gracefully (both for confidentiality and authenticity) with respect to the maximal number of nonce repetitions (i.e., repeating nonces only a few times does not affect the security bound much, and the

same nonce must be repeated close to $2^{n/2}$ times for an adversary to have noticeable advantage). This is very different from OCB3 and AES-GCM for which a single nonce reuse breaks confidentiality and allows universal forgeries. In comparison, COLM [3] only provides the weaker notion of *online* nonce-misuse resistance [28], while AEZ [33] provides nonce-misuse resistance (and even the stronger notion of robustness, which guarantees security against release of unverified plaintext [4]) but only up to the birthday bound. Compared to AES-GCM-SIV [32], which is also nonce-misuse resistant, Deoxys-II provides a larger security margin. For example, when encrypting 2^{32} messages of 64 KB each with the same nonce, the attacker gets an advantage of about 2^{-41} against AES-GCM-SIV versus only 2^{-51} for Deoxys-II.

5 Design Rationale for Deoxys-TBC

The starting point of our design is to provide a sound ad-hoc AES-based tweakable block cipher that has full security. Having such a primitive is beneficial for many authenticated encryption modes that are secure beyond the birthday bound, but lose this feature when instantiated with the current constructions that use a permutation or a cipher as a black box and surround it with addition of words produced by a finite field multiplications (beyond-birthday security authenticated encryption modes that use a block cipher remain quite slow). Therefore, designing a secure tweakable block cipher will enable us to reach full 128-bit security for both confidentiality and authenticity.

5.1 Details for the STK Construction

Designing a secure round function for block ciphers has become a fairly easy task: an S-box layer and a diffusion layer based on MDS code immediately provide good security margin against differential and linear attacks even when the number of rounds in the cipher is small. The problem when designing ciphers, however, lies in how to choose the key schedule – for the cipher to be secure the number of rounds has often to be very large. The complexity of this task increases manifold if the key size is larger and if the key schedule is supposed to be simple (no non-linear operations, and as few linear operations as possible).

We provide a solution to tackle the above two main points in the form of the STK construction. This construction gives a simple key schedule for arbitrary length keys and with an additional checks on related-key attacks, ensures that the cipher is secure. The number of total rounds in the cipher is kept fairly small because of a special trick we use in the key schedule. We split the master key on equal key sizes, each with its own (but similar to the other) simple schedule that produces subkeys that are added simultaneously to the state. Due to the similarity of the schedules, and the use of simple linear layers to differentiate them, we can control the number of difference cancellations happening in the subkeys in a related-key attack. Thus, the security against these type of attacks can be proven with a number of rounds that is not necessarily very high.

In detail, we let TK-p denote the cipher where the master key is p times larger than the state (and thus is divided into p keys). In Deoxys, we work only with TK-2, and TK-3, but the same strategy would work when $p > 3$. Let us choose an arbitrary position of a byte at the beginning of each of the p key schedules. For instance, we fix the position $(1, 1)$ and we investigate how the p bytes at this position at the beginning of the p key schedules, change during the production of the subtweakeys. What is interesting is that as all key schedules apply the same permutation h, the initial p bytes will always be XORed at the same position in the subtweakeys (taking into account the definition of the permutation h we can see that the positions through the rounds change as: (1,1) in the first, (2,1) in the second, (3,2) in the third, (4,4) in the fourth, etc). From the initial p-tuple of byte values $\mathbf{x} = [x_1^0, \ldots, x_p^0]$, the STK key schedule (which can be seen as p similar key schedules that differ only in the linear layers used to update them) produces r tuples $[x_1^1, \ldots, x_p^1], \ldots, [x_1^r, \ldots, x_p^r]$, such that $x_i^{k+1} = L_j(x_i^k)$, where L_i represents the linear layer that updates the i-th word of the schedule. All of them are integrated to the internal state by considering the $r + 1$ XOR values $\bigoplus_{i=1}^{p} x_i^k$, for $0 \leq k \leq r$.

The goal was then to choose the linear layers L_i such that the number of difference cancellations in the sequence $\bigoplus_{i=1}^{p} x_i^k$ is minimized. By choosing the simple LFSRs given in Section 3 (and already used in the lightweight block cipher SKINNY [5]), we have checked with a computer program that cancellation of values (and differences in general as the key schedule is linear) in a chosen byte of TK-p cannot occur more than $p-1$ times over 15 consecutive subkeys. For TK-2, this means that the cumulative difference coming from the p tweakey words can be canceled only once by XOR for 15 consecutive subkeys. For TK-3, this cancellation event can happen twice for 15 consecutive subkeys. We note that for the case of Deoxys-TBC-384, the number of rounds for which we can control the difference cancellations (15) is slightly smaller than the total number of rounds of the cipher (16). However, this has no impact since the security proofs we will aim only apply to a rather small number of rounds $r' < 16$.

5.2 The Choice of Permutation h

The above strategy of designing the key schedule is only the first step that ensures the schedule is not trivially insecure against related-key attacks (and that does not require a huge number of rounds to make the cipher secure). The step that follows is the choice of the bytes position permutation h. With a random search we found a suitable permutation h that provides security against related-key attacks in the relatively small number of rounds. The security of this permutation was provided by the tools specified in Section 6.

Further we discuss the question of optimality of the found h. There are $16! \approx 2^{42}$ potential candidates for h. Despite the fact that this number may not seem too large, running a full brute force search is entirely infeasible because the actual per-instance security check is computationally expensive (a few minutes per permutation). For this reason, we can run only impartial search. Instead of a simple random search, we use more advanced search strategies based on

meta-heuristics, which have been used for optimization of symmetric-key designs in [53].

Meta-heuristics are used to optimize black-box objective functions iteratively (they not necessarily find the global optimum). In our case, the input to the objective function is permutation h, while the output is the number of active S-boxes in the best related-key differential attack. The objective function is computable by solving the MILP [51] that corresponds to the chosen h. Note, we need to find h that provides good security levels simultaneously for both `Deoxys-TBC-256` and `Deoxys-TBC-384`. Therefore, our objective function will be the sum of the active S-boxes for the two ciphers. As discussed in Section 6, in each of them, the number of active S-boxes need to be at least 22. For comparison, in the original permutation h, these numbers are 23 and 24 active S-boxes, achieved with 9-round `Deoxys-TBC-256` and 11-round `Deoxys-TBC-384`, respectively (thus the value of the objective function for this h is 47). All of our results reported further are obtained by running two types of meta-heuristics (simulated annealing and genetic algorithm), each on a single core for several days. We state the results for the best found permutations only.

First, we ran the search for better h permutations for 9-round `Deoxys-TBC-256` and 11-round `Deoxys-TBC-384`. We found a few permutations that achieved 48 active S-boxes (23 +25), against the 47 in the original h. So, the security margin of the current round-reduced ciphers can be improved – `Deoxys-TBC-256` can have one more active S-box in the best trail.

Second, we tried to reduce the current minimal number of rounds to achieve security against related-key differential trails. More precisely, we searched for better permutations in two scenarios: 8-round `Deoxys-TBC-256` combined with 11-round `Deoxys-TBC-384`, and 9-round `Deoxys-TBC-256` combined with 10-round `Deoxys-TBC-384` . In the first case, our search did not produce any permutation that has the required 22 active S-boxes for both of the ciphers (the failure occurred because no permutation was found to have 22 active in 8-round `Deoxys-TBC-256`, rather only 19). Similarly, in the second the best found permutation could not satisfy 22 active for 10-round `Deoxys-TBC-384` (only 21). Therefore, we did not find a way to further reduce the rounds of the ciphers while maintaining appropriate security margin.

To summarize, our extended meta-heuristcs search shows near optimality of the current permutation h – a better permutation will only improve the security bound of one of the block ciphers by a single active S-box against related-key differential attacks. This improvement is only with regards to the security. On the other hand, our original permutation h has the advantage of being very easy to implement on some platforms (refer to the next section). Therefore, with this in mind, we conclude that the current h strikes a good balance between security and ease of implementation (by both criteria it is on of the top candidates).

5.3 From Block Cipher to Tweakable Block Cipher

The `STK` construction (with specified permutation h and linear layers L_i) provides only a secure block cipher with an arbitrary length key. However, turning this

block cipher into a tweakable block cipher is trivial: some bits of the master key are announced as tweak, while the remaining bits are kept as secret key bits. As the key and the tweak are treated in the same way in our designs, we give them the general name *tweakey*. From the TK-2 block cipher that in our case has 256-bit key and 128-bit block, we were able to obtain tweakable block ciphers with 128-bit key and 128-bit tweak (called Deoxys-TBC-128). A similar transition was made from the TK-3 block cipher (with 384-bit tweakey) to Deoxys-TBC-256.

During this transition, it is important to note that the security of the cipher against related-key (and now related-tweak) attacks does not drop, even though parts of the original master key become available to the attacker. The reason for this is twofold: first, the key schedule is linear, it never has any active S-boxes; and second, the XOR of all subkeys/subtweaks in each round to the state is secret (as long as one of them is secret), and also the state is secret (thus the attacker cannot reduce the number of active S-boxes by controlling the tweak).

6 Security Analysis of Deoxys-TBC

In the past two decades, the Advanced Encryption Standard (AES) and AES-type ciphers have been the subject of extensive analysis. As a result, the security of these ciphers against the most popular forms of cryptanalysis, the differential and the linear attacks, is well understood in the single-key model. Important progress in AES security analysis has been provided in the past several years, and it involved careful study of the key schedule of AES-type ciphers. In other words, the latest attacks rely on how the key is processed in the rounds of the ciphers. Two such notable examples are the related-key differential attacks [11, 12] and the Meet-in-the-Middle (MITM) attacks [21, 23, 26, 62] on AES.

Our TWEAKEY framework allows a dual view of the whole constructions. The first is as described previously, i.e. in each round a subkey and a subtweak are added to the state. In the second view, however, one can treat the XOR of the subkey and the subtweak as one single subkey called *subtweakey*, which is produced from a more complex key schedule (composition of the original key schedule and tweak schedule). This way the security analysis of TWEAKEY reduces to the security analysis of a block cipher with more complex key schedule, and where one part of the key is secret, and the second is public.

We view the Deoxys tweakey schedule as an improvement over the AES key schedule: not only it is much simpler and more efficient, but it also provides stronger security guarantees against related-key differential cryptanalysis for example. Since its original publication in [36, 37], Deoxys-TBC sustained several third-party cryptanalysis efforts. We summarize here our own analysis and also these third-party advances.

6.1 Differential Cryptanalysis

Designing a SPN cipher resistant against single-key differential attacks is fairly simple and can be done by carefully choosing the diffusion layer (ensuring that

its branch number is high enough). For example, in the case of AES, because its diffusion matrix has a branching number of 5, one can prove at least 25 differentially active S-boxes for four rounds of AES in the single-key model. Assuming that each of the active S-boxes can reach the maximal differential probability of the AES S-box $p_{max} = 2^{-6}$, one directly deduce that four rounds already provide sufficient protection against simple differential cryptanalysis in the single-key model.

Since our tweakable block cipher Deoxys-TBC is directly built on the AES round function and since the number of active S-boxes is independent of the key schedule in the single-key model, the very same analysis can be applied to Deoxys-TBC. Thus, four rounds of Deoxys-TBC already provide sufficient protection against simple differential cryptanalysis in the single-key model.

While the single-key case is straightforward, it is much harder to guarantee resistance against related-key differential attacks and the STK construction is an elegant way to tackle this problem. There exists search algorithms and tools [13, 14, 27, 29, 51, 62] that given a key schedule can return the upper bound on the probability of the best related-key differential characteristics, and in the case when such a bound is low, practically provide and prove the resistance against related-key differential attacks. The STK construction greatly facilitates the applicability of these tools and we used precisely these algorithms in our security analysis against related-key attacks.

These search tools have been designed to look for related-key characteristics, however, we allow the adversary to operate in a stronger setting of related-key and possibly related-tweak (or both at the same time) attacks. Nonetheless, we can accommodate and modify the tools to search for such characteristics. Although the modification can be done easily, the feasibility (expressed in the time complexity required the search algorithm to finish) is the real problem. To cope with this, we use several different tools, each chosen to provide the probability bounds in the shortest time. More precisely, we alternate between the search algorithm based on Matsui's approach [13], split approach [14], and extended split approach [27]. We omit the details on how these search algorithms operate due to their complexity, and further, provide only the final results produced by the tools. We give in the first line of Table 5 these results for our two tweakable block ciphers Deoxys-TBC-256 and Deoxys-TBC-384.

Our bounds have later been improved using MILP search tools [17] (see the second line of Table 5). Most of these bounds are not tight and one can expect even more active S-boxes in practice. The third line provides bounds on the number of active S-boxes using a reasonable assumption based on the amount of freedom degrees available to the attacker. Indeed, in order to reach the bounds of the second line, only pure structural constraints have been taken into account, while extra linear constraints might come in play for AES-like ciphers [29] and make impossible most differential paths minimizing the number of active S-boxes. Thus, a reasoning on the freedom degrees available allows to assume that the linear system of extra constraints might not have solutions, and thus render differential paths simply impossible. One can view these third-line bounds as an

Table 5: Original and new lower bounds on the number of active S-boxes for `Deoxys-TBC-256` and `Deoxys-TBC-384`. Linear incompatibility bounds require an extra assumption based on the freedom degree of the attacker.

`Deoxys-TBC-256`

#rounds	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
original [37]	0	0	1	5	9	12	16	17	-	22	-	-	-	-
simple model [17]	0	0	1	5	9	12	16	19	23	26	29	32	35	38
linear inc. [17]	0	0	1	5	10	14	18	22	27	31	35	40	44	48

`Deoxys-TBC-384`

#rounds	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
original [37]	0	0	0	1	4	8	-	-	-	-	-	22	-	-	-	-
simple model [17]	0	0	0	1	4	8	10	14	18	21	24	28	31	35	37	45
linear inc. [17]	0	0	0	1	5	9	13	18	22	27	31	35	40	44	48	52

indication of the actual minimum number of active S-boxes one attacker could hope for.

We assume that each of the active S-boxes can reach the maximal differential probability of the `AES` S-box $p_{max} = 2^{-6}$. Using at least $r = 9$ rounds for `Deoxys-TBC-256`, the number of active S-boxes is lower-bounded by 23, meaning that the probability of the associated differential characteristic is upper-bounded by $2^{-6\times 23} = 2^{-138}$. Thus, such characteristics cannot be exploited due to the state size of 128 bits (the attacker cannot construct more than 2^{128} plaintext pairs to start the attack). A similar reasoning with linear constraints bounds indicates that $r = 8$ might actually be sufficient.

Using at least $r = 11$ rounds for `Deoxys-TBC-384`, the number of active S-boxes is lower-bounded by 24, meaning that the probability of the associated differential characteristic is upper-bounded by $2^{-6\times 24} = 2^{-144}$ and such characteristics cannot be exploited due to the state size of 128 bits. A similar reasoning with linear constraints bounds indicates that $r = 9$ might actually be sufficient.

Thus, `Deoxys-TBC-256` and `Deoxys-TBC-384` have a security margin of at least five and seven rounds respectively and thus are highly resistant against related-key related-tweak attacks.

6.2 Linear Cryptanalysis

Similarly to the differential cryptanalysis case, the security guarantees of `AES` with regards to linear cryptanalysis in the single-key model directly translate to

Deoxys-TBC. Therefore, we can easily prove that four rounds of Deoxys-TBC-256 or Deoxys-TBC-384 already provide sufficient protection against simple linear cryptanalysis in the single-key model.

Moreover, since there is no cancellation of linearly active S-boxes in the related-key model [42], the analysis of single-key directly translates to the related-key model as well. Therefore, four rounds already provide sufficient protection against simple linear cryptanalysis also in the related-key model.

One might argue that the linearity of the Deoxys-TBC-256 or Deoxys-TBC-384 tweakey schedule will help linear cryptanalysis. However, we emphasize that most analysis of AES with regards to linear cryptanalysis have been done assuming that the subkeys are independent. In addition, unlike for AES, the tweakey schedule of Deoxys-TBC-256 or Deoxys-TBC-384 has been chosen to maximize the number of active S-boxes and the key bytes diffusion. This will very likely render its cryptanalysis even harder for an attacker.

6.3 Boomerang Attacks and Variants

A boomerang attack [65] regards the target cipher as a composition of two sub-ciphers E_0 and E_1: the first sub-cipher is supposed to have a differential $\alpha \rightarrow \beta$, and the second one to have a differential $\gamma \rightarrow \delta$, with probabilities p and q, respectively. The basic boomerang attack requires an adaptive chosen plaintext/ciphertext scenario, and plaintext pairs result in a right quartet with probability p^2q^2. A variant working in a chosen-plaintext scenario, so-called amplified boomerang attack, can produce a right quartet with probability $p^2q^2 2^{-n}$ [38]. Further, it was pointed out in [8, 9] that any value of β and γ is allowed as long as $\beta \neq \gamma$. As a result, the probability of the right quartet is increased to $2^{-n}\hat{p}^2\hat{q}^2$, where $\hat{p} = \sqrt{\Sigma_i \mathrm{Pr}^2(\alpha \rightarrow \beta_i)}$ and $\hat{q} = \sqrt{\Sigma_j \mathrm{Pr}^2(\gamma_j \rightarrow \delta)}$, and this variant is called rectangle attack.

As one can observe in Table 5, for Deoxys-TBC-256 or Deoxys-TBC-384 the amount of minimum active S-boxes does not grow immediately in the related-key model and thus boomerang attacks and variants seem to be a good candidate to attack them. Indeed, in a boomerang attack, the attacker will look for a potentially short but highly probable differential path for the primitive analysed. The shortness of the path will be compensated by the fact it will be used only for roughly half of the rounds attacked, while the high probability will ensure that the final boomerang differential characteristic will have reasonable success probability.

A first attempt was performed on Deoxys-TBC-256 an Deoxys-TBC-384 in [17], where (very costly) related-key boomerang characteristics were produced for up to 11 rounds of Deoxys-TBC-256 and 13 rounds of Deoxys-TBC-384, see Table 6.

These related-key characteristics have then been turned into related-key rectangle attacks (simple distinguisher or key recovery) on reduced versions of Deoxys-TBC-256 and Deoxys-TBC-384 (see Table 7). The authors even considered beyond-full codebook scenario, which can make sense for tweakable block

Table 6: Related-key boomerang characteristics on `Deoxys-TBC-256` and `Deoxys-TBC-384` [17]. Notation $\hat{p}$ and $\hat{q}$ depicts here improved probabilities due to boomerang switching techniques. The number of rounds corresponds to R_1 (top differential characteristic) R_2 (bottom differential characteristic).

`Deoxys-TBC-256`				`Deoxys-TBC-384`			
#rounds	(R_1, R_2)	pq	$\hat{p}^2\hat{q}^2$	**#rounds**	(R_1, R_2)	pq	$\hat{p}^2\hat{q}^2$
8	(4, 4)	2^{-36}	2^{-72}	10	(5, 5)	2^{-24}	2^{-42}
9	(5, 4)	2^{-61}	2^{-122}	11	(6, 5)	2^{-60}	2^{-120}
10	(5, 5)	2^{-106}	2^{-212}	12	(6, 6)	2^{-98}	2^{-196}
11	(6, 5)	2^{-136}	2^{-265}	13	(7, 6)	2^{-134}	2^{-268}

ciphers in extreme scenario as the tweak input can provide more data to the attacker than just the cipher domain size. Overall, 10 and 12 rounds could be reached for `Deoxys-TBC-256` and `Deoxys-TBC-384` respectively when used with a 128-bit and 256-bit key respectively, or 11 and 14 rounds respectively by increasing the key size in the tweakey state.

Interestingly, the authors analysed the case where the tweakable block ciphers are placed inside the mode and only reduced-round `Deoxys-I` could be attacked (9 rounds for `Deoxys-I-128` and 12 rounds for `Deoxys-I-256`, see Table 7). It seems that these related-key boomerang attacks are much harder to translate into threats for the `Deoxys-II` mode.

These results were slightly improved in [18], where the authors introduced a new metric tool, the Boomerang Connectivity Table (BCT), to analyze more systematically and more precisely the possible interactions between the top and bottom differential characteristics in a boomerang attack. More precisely, existing boomerang distinguishers [17] on 8, 9 and 10 rounds of `Deoxys-TBC-384` with probability 2^{-6}, 2^{-18}, 2^{-42} respectively were each improved by a factor $2^{0.6}$. Another improvement came from [66] where a factor $2^{1.6}$ was gained on the 9-round boomerang distinguisher probability for `Deoxys-TBC-256`.

In [61], by optimizing attack procedures, the authors managed to reduce the complexities of 8- and 9-round related-tweakey boomerang distinguishers against `Deoxys-TBC-256` to 2^{28} and 2^{98} computations, respectively, whereas the previous attacks require 2^{74} and 2^{124} respectively. These distinguishers were extended to 9- and 10-round boomerang key-recovery attacks with complexity 2^{112} and 2^{170}.

6.4 Meet-in-the-Middle Attacks

Additionally, we scrutinized the resistance of our design with regards to the advanced meet-in-the-middle attack on `AES` conducted in [23]. Indeed, this attack strongly relies on the `AES` key schedule to propagate linear equations in the MITM strategy to spare some guesses in both the offline and online phases. As

Table 7: Related-key rectangle attack results from [17, 61] on reduced-round versions of `Deoxys-TBC-256` and `Deoxys-TBC-384` (top table), as well as on reduced-round versions of the AEAD schemes `Deoxys-I-128` and `Deoxys-I-256` (bottom table).

`Deoxys` **Internal Primitives**

	number of rounds	tweak size	key size	time	data	memory	attack type	ref.
`Deoxys-TBC-256`	8/14	128	128	2^{74}	2^{74}	-	distinguisher	[17]
		128	128	2^{28}	2^{28}	2^{27}	distinguisher	[61]
	9/14	128	128	2^{124}	2^{124}	-	distinguisher	[17]
		128	128	2^{98}	2^{98}	2^{17}	distinguisher	[61]
		128	128	2^{118}	2^{117}	2^{117}	key-recovery	[17]
		128	128	2^{112}	2^{98}	2^{17}	key-recovery	[61]
	10/14	$t < 52$	$k > 204$	2^{204}	$2^{127.58}$	$2^{127.58}$	key-recovery	[17]
		$t < 86$	$k > 170$	2^{170}	2^{170}	2^{17}	key-recovery	[61]
		$t < 86$	$k > 170$	2^{170}	2^{98}	2^{98}	key-recovery	[61]
		128	128	$2^{109.1}$	$2^{98.4}$	2^{88}	key-recovery	[70]
	11/14	$t < 4$	$k > 252$	$2^{249.9}$	$2^{122.1}$	$2^{128.2}$	key-recovery	[70]
`Deoxys-TBC-384`	10/16	128	256	2^{44}	2^{44}	-	distinguisher	[17]
		128	256	2^{22}	2^{22}	2^{17}	distinguisher	[61]
	11/16	128	256	2^{122}	2^{122}	-	distinguisher	[17]
		128	256	2^{100}	2^{100}	2^{17}	distinguisher	[61]
	12/16	128	256	2^{127}	2^{127}	2^{125}	key-recovery	[17]
		128	256	2^{148}	2^{148}	2^{17}	key-recovery	[61]
		128	256	2^{148}	2^{100}	2^{100}	key-recovery	[61]
		256	128	2^{98}	2^{98}	2^{64}	key-recovery	[70]
		128	256	2^{208}	2^{115}	2^{113}	key-recovery	[70]
	13/16	$t < 114$	$k > 270$	2^{270}	2^{127}	2^{144}	key-recovery	[17]
		128	256	$2^{191.3}$	2^{125}	2^{136}	key-recovery	[70]
		128	256	$2^{186.7}$	$2^{125.2}$	2^{136}	key-recovery	[71]
	14/16	$t < 98$	$k > 286$	$2^{286.2}$	2^{127}	2^{136}	key-recovery	[70]
		$t < 102$	$k > 282$	$2^{282.7}$	$2^{125.2}$	2^{136}	key-recovery	[71]

`Deoxys` **AE Schemes**

`Deoxys-I-128`	9/14	-	128	2^{118}	2^{117}	2^{117}	RK rectangle	[17]
	10/14	-	128	$2^{114.2}$	$2^{114.2}$	$2^{112.2}$	RK rectangle	[70]
`Deoxys-I-256`	12/16	-	256	2^{236}	2^{126}	2^{124}	RK rectangle	[17]
	12/16	-	256	2^{208}	2^{115}	2^{113}	RK rectangle	[70]
	13/16	-	256	$2^{186.7}$	$2^{125.2}$	2^{136}	RK rectangle	[71]

the design we propose introduces a new tweakey schedule, we have analyzed how it interacts with the `AES` round function.

For a given tweak value, `Deoxys-TBC` behaves as the `AES` with a new schedule with partially known values (the subtweakeys) XORed between each round, without additional input values. This tweakey schedule is fully linear as it first applies a byte permutation and then an LFSR on some bytes of the state. In that context, a first analysis shows that the meet-in-the-middle technique from [23]

can attack up to 8 rounds, where the AES key schedule for 128-bit keys stops the attack at 7 rounds.

Several years after our first analysis, it seems our estimations were accurate: in [46], the authors applied using meet-in-the-middle attacks managed to reach 8 rounds of Deoxys-TBC-256 with 2^{113} time/data complexity and 2^{97} memory, and 10 rounds of Deoxys-TBC-384 with 2^{228} time, 2^{113} data and 2^{226} memory complexity.

6.5 Impossible Differential Attacks

In [50], the authors proposed impossible differential attacks against reduced-round Deoxys-TBC-256. For a very high complexity (2^{118} computational and data, 2^{114} memory), the authors managed to attack up to 9 rounds of Deoxys-TBC-256 in the related-tweak related-key model. Using the same strategy, they could attack 8 rounds in the related-tweak model ($2^{116.5}$ computational and data, 2^{48} memory).

6.6 Other Attacks

As mentioned earlier in the chapter, the security bound of Deoxys-TBC-256 or Deoxys-TBC-384 against most of the other attacks matches the bounds of AES, i.e. all the attacks that do not exploit the key schedule will have the same success on Deoxys-TBC-256 or Deoxys-TBC-384 as on AES. This gives us a security reduction from AES, however, we note that as our ciphers have more rounds, for these particular attacks their security margin is higher than AES.

Besides the above types of attacks, we encourage to investigate attack vectors that rely on some additional property of the key schedule of Deoxys-TBC-256 or Deoxys-TBC-384. We emphasize that other attack techniques like slide [15], rotational [40] and the internal differential attacks [54] are prevented by the usage of the constants in the key schedule, as done in the AES.

As previously described, since by design there is no distinction between key and tweak material for Deoxys-TBC-256 or Deoxys-TBC-384 (rather the key+tweak inputs are treated as one tweakey input), trivial so-called related-cipher attacks [67] would apply between Deoxys-TBC-256 and Deoxys-TBC-384. As the practical threat coming from this type of attack framework is very limited and can be completely avoided at the operating mode level, we decided not to put different constants RC_i in order to prevent the attack. Moreover, we recall that for the two Deoxys operating modes, the separation between the tweakable block cipher instances is naturally and safely done since the tweak and key sizes are fixed and the placement of key and tweak material is fully determined.

Finally, we note that a possible increment in the number of attacked rounds might happen in the scenario of open-key distinguishers (even though we have not been able to improve the known attacks [22, 31, 35] using this extra tweak input). However, we emphasize that we do not claim any resistance of Deoxys-TBC-256 nor Deoxys-TBC-384 in this attack model.

Table 8: Proved bounds on the minimal number of differential active S-boxes for AES-256 and for Deoxys-TBC-256. Model **SK** denotes the single-key scenario and model **RTK** denotes the related-tweakey scenario where differences can be inserted in the tweakey state.

Cipher	Model	Rounds							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Deoxys-TBC-256	**SK**	1	5	9	25	26	30	34	50
(14 rounds)	**RTK**	0	0	1	5	9	12	16	19
AES-256	**SK**	1	5	9	25	26	30	34	50
(14 rounds)	**RTK**	0	0	1	3	5	5	5	10

6.7 Comparing Deoxys-TBC with AES

The Deoxys-TBC-256 and Deoxys-TBC-384 tweakable block ciphers are directly built upon the AES round function, but their key schedule has been updated. We see this new key schedule as a direct improvement over the AES key schedule. We recall that AES-192 and AES-256 have been shown to be weak against related-key attacks [11, 12], thus indicating that their respecting key schedule is not strong enough against certain type of attacks.

We compare in Table 8 the minimum number of differentially active S-boxes of AES-256 and Deoxys-TBC-256 in the related-key model (we compare these two primitives as they have the same tweakey size). One can see that Deoxys-TBC-256 tweakey schedule guarantees many more active S-boxes than AES-256 (and thus an expected higher resistance against related-key attacks), while being much more efficient. Note that the bounds for Deoxys-TBC-256 are not tight and can probably be further improved.

We finally emphasize that Deoxys-TBC-256 is used in the 128-bit key mode of Deoxys, and thus attacks requiring more than 2^{128} computations are not a concern (in contrary to AES-256). This further increases the security margin Deoxys-TBC-256 provides when compared to AES-256.

7 Security Proof of SCT-2

In this section, we give a self-contained security proof of the SCT-2 mode. The SCT-2 mode relies on the NSIV generic composition method (introduced in [55]) applied to two building blocks, a nonce-based keyed function called NaT (introduced in [19]) and a combined nonce and IV-based encryption scheme called CTRT (introduced in [55]).

7.1 Definitions and Security Notions

Given two sets $\mathcal{X}$ and $\mathcal{Y}$, the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$ is denoted $\mathsf{Func}(\mathcal{X}, \mathcal{Y})$. A function $F \in \mathsf{Func}(\mathcal{X}, \mathcal{Y})$ is said *regular* if all $Y \in \mathcal{Y}$ have the same number of preimages by F (this obviously requires $|\mathcal{X}|$ to be a multiple of $|\mathcal{Y}|$).

Let $\varepsilon > 0$, and let $H : \mathcal{K}_h \times \mathcal{X} \to \mathcal{Y}$ be a keyed function for three non-empty sets $\mathcal{K}_h$, $\mathcal{X}$, and $\mathcal{Y}$ with $\mathcal{K}_h$ finite. H is said to be *ε-almost universal (ε-AU)* if for any distinct X and $X' \in \mathcal{X}$,

$$\Pr\left[K_h \leftarrow_{\$} \mathcal{K}_h : H_{K_h}(X) = H_{K_h}(X')\right] \leq \varepsilon.$$

`SCT-2` follows the `NSIV` generic composition method, which relies on two building blocks, a nonce-based pseudorandom MAC (nPRM) and a nivE scheme, which we define below.

nPRM Schemes. A nonce-based keyed function is a function $F : \mathcal{K} \times \mathcal{N} \times \mathcal{D} \to \mathcal{Y}$, where $\mathcal{K}$ is the key space, $\mathcal{N}$ the nonce space, $\mathcal{D}$ the domain and $\mathcal{Y}$ the range. We introduce a new security notion called nonce-based pseudorandom MAC (nPRM) which conveniently combines the PRF and MAC security notions and simplifies the security analysis of the `NSIV` composition method.

Definition 3 (nPRM-security). *Let $F : \mathcal{K} \times \mathcal{N} \times \mathcal{D} \to \mathcal{Y}$ be a nonce-based keyed function, and let us write $F_K(N, D)$ for $F(K, N, D)$. Let A be an adversary with oracle access to two oracles, the first oracle being a function from $\mathcal{N} \times \mathcal{D}$ to $\mathcal{Y}$, the second oracle with inputs in $\mathcal{N} \times \mathcal{D} \times \mathcal{Y}$ and outputs in $\{\top, \bot\}$. The advantage of A against the nPRM-security of F is defined as*

$$\mathbf{Adv}_F^{\mathtt{nPRM}}(\mathsf{A}) = \Big|\Pr\left[K \leftarrow_{\$} \mathcal{K} : \mathsf{A}^{F_K, \mathsf{Ver}_K} = 1\right] - \Pr\left[\rho \leftarrow_{\$} \mathsf{Func}(\mathcal{N} \times \mathcal{D}, \mathcal{Y}) : \mathsf{A}^{\rho, \mathsf{Rej}} = 1\right]\Big|,$$

where Ver_K is an oracle which takes as input a triple $(N, D, \mathsf{tag}) \in \mathcal{N} \times \mathcal{D} \times \mathcal{Y}$ and returns $\top$ if $F_K(N, D) = \mathsf{tag}$ and $\bot$ otherwise and Rej is an oracle which always returns $\bot$. The adversary is not allowed to ask a verification query (N, D, tag) if a previous query (N, D) to F_K returned tag.

Note that a nonce-based keyed function might be a PRF against an adversary repeating any nonce at most μ times but fail to be an nPRM against an adversary repeating any nonce at most μ times in its left-oracle queries. Think for example of a function F which depends only on the nonce: even if it is a PRF against nonce-respecting adversaries, it is *not* a secure nPRM against a nonce-respecting adversary: simply query the left oracle on some (N, D) and then the second oracle on (N, D') for $D' \neq D$; it will return $\top$ in the (F_K, Ver_K) world and $\bot$ in the (ρ, Rej) world.

nivE Schemes. Most existing encryption schemes are either nonce-based [59] or IV-based [6], i.e., they employ an externally provided value which either should not repeat (nonce), or should be selected uniformly at random (IV). (See

also [52].) The notion of combined nonce- and IV-based encryption scheme (*nivE* for short) was introduced in [55].

Syntactically, a nivE scheme is a tuple $\Pi = (\mathcal{K}, \mathcal{N}, \mathcal{IV}, \mathcal{M}, \mathcal{C}, \mathsf{Enc}, \mathsf{Dec})$ where $\mathcal{K}$, $\mathcal{N}$, $\mathcal{IV}$, $\mathcal{M}$, and $\mathcal{C}$ are non-empty sets of bit strings with $\mathcal{K}$, $\mathcal{N}$, and $\mathcal{IV}$ finite and Enc and Dec are deterministic algorithms. The encryption algorithm Enc takes as input a key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, an initial value $IV \in \mathcal{IV}$, and a message $M \in \mathcal{M}$, and outputs a ciphertext $C \in \mathcal{C}$ (we assume that Enc returns $\perp$ if one of the inputs is not in the intended set). The decryption algorithm Dec takes as input a key $K \in \mathcal{K}$, a nonce $N \in \mathcal{N}$, an initial value $IV \in \mathcal{IV}$, and a ciphertext $C \in \mathcal{C}$, and outputs either a message $M \in \mathcal{M}$, or a special symbol $\perp$. We require that

$$\mathsf{Dec}(K, N, IV, \mathsf{Enc}(K, N, IV, M)) = M$$

for all tuples $(K, N, IV, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{IV} \times \mathcal{M}$. We also require that if $\mathcal{M}$ contains a bit string of length m, then it contains all bit strings of length m, and that $|\mathsf{Enc}(K, N, IV, M)| = |M|$ for all $(K, N, IV, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{IV} \times \mathcal{M}$.

We let $\mathsf{Enc}^{\$}$ denote the probabilistic algorithm which on input $(K, N, M) \in \mathcal{K} \times \mathcal{N} \times \mathcal{M}$ internally generates a uniformly random $IV \leftarrow_{\$} \mathcal{IV}$, computes $C = \mathsf{Enc}(K, N, IV, M)$, and outputs $(IV, C) \in \mathcal{IV} \times \mathcal{C}$. We write $\mathsf{Enc}_K(N, IV, M)$ for $\mathsf{Enc}(K, N, IV, M)$ and $\mathsf{Enc}^{\$}_K(N, M)$ for $\mathsf{Enc}^{\$}(K, N, M)$. The security of a nivE scheme is defined as follows.

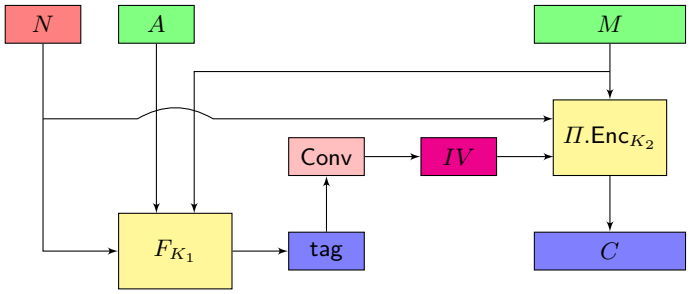
Definition 4 (nivE-security). *Let $\Pi = (\mathcal{K}, \mathcal{N}, \mathcal{IV}, \mathcal{M}, \mathcal{C}, \mathsf{Enc}, \mathsf{Dec})$ be a nivE scheme. The advantage of an adversary A against the nivE-security of Π is defined as*

$$\mathbf{Adv}^{\mathrm{nivE}}_{\Pi}(\mathsf{A}) = \left| \Pr\left[K \leftarrow_{\$} \mathcal{K} : \mathsf{A}^{\Pi.\mathsf{Enc}^{\$}_K} = 1\right] - \Pr\left[\mathsf{A}^{\mathsf{Rand}} = 1\right] \right|,$$

where Rand is an oracle which on input $(N, M) \in \mathcal{N} \times \mathcal{M}$ outputs a uniformly random pair $(IV, C) \leftarrow_{\$} \mathcal{IV} \times \{0,1\}^{|M|}$.

Adversary Characteristics. In all this section, given some implicit parameter n, a $(q, \mu, \ell, \sigma, t)$-adversary against a nonce-based scheme is an adversary:

- which makes at most q oracle queries; when the adversary has access to two oracles (i.e., when attacking the nPRM-security of a keyed function or a nAE scheme), this means q queries in total to both oracles;
- which uses any nonce at most μ times throughout its queries ($\mu = 1$ for a nonce-respecting adversary); when the adversary has access to two oracles, this only applies to queries to its first oracle (function or encryption oracle);
- such that the length of any of its queries (nonce excluded) is at most ℓ blocks of n bits; for a keyed function with domain $\mathcal{D} = \mathcal{A} \times \mathcal{M}$ or a nAE scheme, this means that *both* the AD length and the message length of any query is at most ℓ blocks of n bits;
- such that the total length of all its queries (nonce excluded) is at most σ blocks of n bits; for a keyed function with domain $\mathcal{D} = \mathcal{A} \times \mathcal{M}$ or a nAE scheme, this means the sum of the AD and the message length over all queries;
- which runs in time at most t.




```
1  algorithm NSIV[F, Π].Enc_{K1,K2}(N, A, M)
2    tag := F_{K1}(N, A, M)
3    IV := Conv(tag)
4    C := Π.Enc_{K2}(N, IV, M)
5    return (C, tag)
6
7  algorithm NSIV[F, Π].Dec_{K1,K2}(N, A, C, tag)
8    IV := Conv(tag)
9    M := Π.Dec_{K2}(N, IV, C)
10   tag' := F_{K1}(N, A, M)
11   if tag' = tag then return M else return ⊥
```

Fig. 8: The NSIV construction, defining a nAE scheme from a nonce-based keyed function $F : \mathcal{K}_1 \times \mathcal{N} \times \mathcal{D} \to \mathcal{Y}$ where $\mathcal{D} = \mathcal{A} \times \mathcal{M}$ and a nivE scheme $\Pi = (\mathcal{K}_2, \mathcal{N}, \mathcal{IV}, \mathcal{M}, \mathcal{C}, \mathsf{Enc}, \mathsf{Dec})$. Function Conv is a regular function from $\mathcal{Y}$ to $\mathcal{IV}$.

7.2 The NSIV Composition Method

We first describe a generic composition method named NSIV, which defines a nAE scheme from a nonce-based keyed function and a nivE scheme. The NSIV construction results from a small (but important from a security viewpoint) modification to the (generic) SIV construction [60]. While in SIV the encryption part is purely IV-based, NSIV relies on a combined nonce- and IV-based encryption (nivE) scheme, the nonce being used as input *both* to the keyed function and the nivE scheme. This is the only difference with SIV, where the nonce is only given as input to the keyed function.

More formally, let F be a nonce-based keyed function with key space $\mathcal{K}_1$, nonce space $\mathcal{N}$, domain $\mathcal{D} = \mathcal{A} \times \mathcal{M}$, and range $\mathcal{Y} = \{0,1\}^\tau$, and $\Pi = (\mathcal{K}_2, \mathcal{N}, \mathcal{IV}, \mathcal{M}, \mathcal{C}, \mathsf{Enc}, \mathsf{Dec})$ be a nivE scheme. Fix a regular function $\mathsf{Conv} : \mathcal{Y} \to \mathcal{IV}$. We define the nAE scheme $\mathtt{NSIV}[F, \Pi] = (\mathcal{K}, \mathcal{N}, \mathcal{A}, \mathcal{M}, \mathcal{C}, \mathsf{Enc}, \mathsf{Dec})$ with key-space $\mathcal{K} = \mathcal{K}_1 \times \mathcal{K}_2$ as specified in Fig. 8.

The security of $\mathtt{NSIV}[F, \Pi]$ is given by Theorem 1 below. We let $t_\Pi(\sigma)$ denote an upper bound on the time needed for computing $\Pi.\mathsf{Enc}$ or $\Pi.\mathsf{Dec}$ on inputs of total message length at most σ blocks of n bits, and we assume that computing $\mathsf{Conv}(\mathsf{tag})$ or sampling uniformly from $\mathsf{Conv}^{-1}(IV)$ takes negligible time for any $\mathsf{tag} \in \mathcal{Y}$ and $IV \in \mathcal{IV}$.

Theorem 1 (Security of NSIV). *Let $F : \mathcal{K}_1 \times \mathcal{N} \times \mathcal{D} \to \mathcal{Y}$, where $\mathcal{D} = \mathcal{A} \times \mathcal{M}$, be a nonce-based keyed function, $\Pi = (\mathcal{K}_2, \mathcal{N}, \mathcal{IV}, \mathcal{M}, \mathcal{C}, \mathsf{Enc}, \mathsf{Dec})$ be a nivE scheme, and $\mathsf{Conv} : \mathcal{Y} \to \mathcal{IV}$ be a regular function. Let A be a $(q, \mu, \ell, \sigma, t)$-adversary against $\mathtt{NSIV}[F, \Pi]$. Then, letting $t' = t + t_\Pi(\sigma)$, there exists a $(q, \mu, \ell, \sigma, t')$-adversary B against the nPRM-security of F and a $(q, \mu, \ell, \sigma, t')$-adversary B' against the nivE-security of Π such that*

$$\mathbf{Adv}^{\mathrm{nAE}}_{\mathtt{NSIV}[F,\Pi]}(\mathsf{A}) \le \mathbf{Adv}^{\mathtt{nPRM}}_{F}(\mathsf{B}) + \mathbf{Adv}^{\mathrm{nivE}}_{\Pi}(\mathsf{B}').$$

Proof. Let $\widetilde{\Pi} = \mathtt{NSIV}[F, \Pi]$ and let A be a $(q, \mu, \ell, \sigma, t)$-adversary against $\widetilde{\Pi}$. Let

$$\begin{aligned} W_1 &= (\widetilde{\Pi}.\mathsf{Enc}_{K_1,K_2}, \widetilde{\Pi}.\mathsf{Dec}_{K_1,K_2}), \\ W_3 &= (\mathsf{Rand}, \mathsf{Rej}) \end{aligned}$$

denote the two worlds that A must tell apart, where $\mathsf{Rand}(N, A, M)$ returns a uniformly random element in $\{0,1\}^{|M|} \times \mathcal{Y}$ and Rej always returns $\perp$. Then

$$\mathbf{Adv}^{\mathrm{nAE}}_{\mathtt{NSIV}[F,\Pi]}(\mathsf{A}) = \left|\Pr\left[\mathsf{A}^{W_1} = 1\right] - \Pr\left[\mathsf{A}^{W_3} = 1\right]\right|. \tag{1}$$

We introduce an intermediate world W_2 where in the encryption oracle we replace F_{K_1} by a uniformly random function $\rho : \mathcal{N} \times \mathcal{D} \to \mathcal{Y}$ and the decryption oracle always returns $\perp$. Formally, letting $\widetilde{\Pi}' = \mathtt{NSIV}[\mathsf{Func}(\mathcal{N} \times \mathcal{D}, \mathcal{Y}), \Pi]$, we define

$$W_2 = (\widetilde{\Pi}'.\mathsf{Enc}_{\rho,K_2}, \mathsf{Rej}).$$

We first consider A's advantage in distinguishing W_1 from W_2. Consider the adversary B against the nPRM-security of F working as follows. Let $(G, V) \in \{(F_{K_1}, \mathsf{Ver}_{K_1}), (\rho, \mathsf{Rej})\}$ be the pair of oracles to which B has access. Adversary B picks a random key $K_2 \leftarrow_\$ \mathcal{K}_2$ and runs A. When A makes an encryption query (N, A, M), B queries $\mathsf{tag} := G(N, A, M)$, computes $IV := \mathsf{Conv}(\mathsf{tag})$ and $C := \Pi.\mathsf{Enc}_{K_2}(N, IV, M)$, and returns (C, tag). When A makes a decryption query (N, A, C, tag), B computes $IV := \mathsf{Conv}(\mathsf{tag})$ and $M := \Pi.\mathsf{Dec}_{K_2}(N, IV, C)$, and queries $V((N, A, M), \mathsf{tag})$; if V returns $\top$ then B returns M, otherwise it returns $\perp$. When A halts and outputs some bit, B outputs the same bit. One can check that when $(G, V) = (F_{K_1}, \mathsf{Ver}_{K_1})$ then B perfectly simulates W_1, whereas when $(G, V) = (\rho, \mathsf{Rej})$ then B perfectly simulates W_2. Hence,

$$\left|\Pr\left[\mathsf{A}^{W_1} = 1\right] - \Pr\left[\mathsf{A}^{W_2} = 1\right]\right| \le \mathbf{Adv}^{\mathtt{nPRM}}_{F}(\mathsf{B}). \tag{2}$$

Moreover, it is easy to see that B is a $(q, \mu, \ell, \sigma, t')$-adversary.

We then consider A's advantage in distinguishing W_2 from W_3. Consider the adversary B' against the nivE-security of Π working as follows. Let $G \in$

```
algorithm CTRT[E].Enc_K(N, IV, M)
  ℓ := |M|/n
  parse M as M_1 ‖ ··· ‖ M_ℓ, with |M_1| = ... = |M_{ℓ−1}| = n and 1 ≤ |M_ℓ| ≤ n
  for i = 1 to ℓ do
    C_i := M_i ⊕ E_K^{1,IV}(N)
    IV := Inc(IV)
  return C_1 ‖ C_2 ‖ ··· ‖ C_ℓ
```

Fig. 9: Definition of the CTRT mode, using a TBC $E \in \mathsf{TBC}(\mathcal{K}, \mathcal{T}', \mathcal{X})$ with $\mathcal{T}' = \{1\} \times \mathcal{T}$ and $\mathcal{X} = \{0,1\}^n$.

$\{\Pi.\mathsf{Enc}^{\$}_{K_2}, \mathsf{Rand}'\}$ be the oracle to which B' has access, where $\mathsf{Rand}'(N, M)$ returns a random string of length $|\Pi.\mathsf{Enc}^{\$}_{K_2}(N, M)|$. Adversary B' runs A. When A asks an encryption query (N, A, M), B' queries $G(N, M)$, thereby obtaining an answer (IV, C). It samples uniformly at random tag in the set of preimages $\mathsf{Conv}^{-1}(IV)$ and returns (C, tag) to A. When A asks a decryption query, B' simply answers $\perp$. When A halts and outputs some bit, B' outputs the same bit. Note that independently of whether G is $\Pi.\mathsf{Enc}^{\$}_{K_2}$ or Rand', the first element IV in its answers is uniformly random, and hence tag is uniformly random as well since Conv is regular. It follows that B' perfectly simulates W_2 when $G = \Pi.\mathsf{Enc}^{\$}_{K_2}$, and perfectly simulates W_3 when $G = \mathsf{Rand}'$. Hence,

$$\left|\Pr\left[\mathsf{A}^{W_2} = 1\right] - \Pr\left[\mathsf{A}^{W_3} = 1\right]\right| \leq \mathbf{Adv}^{\mathrm{nivE}}_{\Pi}(\mathsf{B}'). \tag{3}$$

Moreover, it is easy to see that B' is a $(q, \mu, \ell, \sigma, t')$-adversary.

Combining Eqs. (1), (2), and (3) concludes the proof. □

7.3 Security of the nivE Scheme CTRT

We now define the CTRT (*CounTeR in Tweak*) mode, turning a tweakable block cipher into a nivE scheme. Let $\mathcal{K}$ and $\mathcal{T}$ be finite non-empty sets, and let $E \in \mathsf{TBC}(\mathcal{K}, \mathcal{T}', \mathcal{X})$ be a tweakable block cipher with key space $\mathcal{K}$, tweak space[9] $\mathcal{T}' = \{1\} \times \mathcal{T}$, and domain $\mathcal{X} = \{0,1\}^n$. Let Inc be a cyclic permutation of $\mathcal{T}$. We construct from E a nivE scheme $\mathtt{CTRT}[E]$ with key space $\mathcal{K}$, nonce space $\mathcal{N} = \mathcal{X} = \{0,1\}^n$, IV space $\mathcal{IV} = \mathcal{T}$, and message space $\mathcal{M} = \{0,1\}^*$ as defined in Fig. 9.

The security of CTRT is given by Theorem 2 below which was proved in [55]. Here $t_{\mathtt{CTRT}}(\sigma)$ is an upper bound on the time needed for computing $\mathtt{CTRT}[E].\mathsf{Enc}_K$ on inputs of total message length at most σ blocks of n bits when calls to E_K cost unit time.

[9] Tweak separation is only required in order to combine CTRT with a PRF also based on E through NSIV.

Theorem 2 (Security of CTRT). *Let $E \in \mathsf{TBC}(\mathcal{K}, \mathcal{T}', \mathcal{X})$ with $\mathcal{X} = \{0,1\}^n$ and $\mathcal{T}' = \{1\} \times \mathcal{T}$. Let A be a $(q, \mu, \ell, \sigma, t)$-adversary against $\mathtt{CTRT}[E]$ with $\ell \leq |\mathcal{T}|$. Then there exists an adversary B against the TPRP-security of E, making at most σ oracle queries and running in time at most $t + t_{\mathtt{CTRT}}(\sigma)$, such that*

$$\mathbf{Adv}^{\mathrm{nivE}}_{\mathtt{CTRT}[E]}(\mathsf{A}) \leq \mathbf{Adv}^{\mathrm{TPRP}}_{E}(\mathsf{B}) + \frac{2(\mu-1)\sigma}{|\mathcal{T}|} + \frac{\sigma^2}{2|\mathcal{X}||\mathcal{T}|}.$$

7.4 Security of the Nonce-Based Keyed Function

At a high-level, the nonce-based keyed function used in `SCT-2` follows the *UHF-then-PRF* paradigm: the AD and the message are hashed with a (TBC-based) universal hash function and a (TBC-based) PRF is applied to the output. The difference here is that the PRF also takes a nonce as the tweak input to the TBC. This high-level construction was proposed in [19] under the name `NaT` (*Nonce-as-Tweak*).

More formally, given a TBC E with key space $\mathcal{K}$, tweak space $\mathcal{T}$, and domain $\mathcal{X} = \{0,1\}^n$ and a keyed hash function H with key space $\mathcal{K}_h$, domain $\mathcal{D}$, and range $\{0,1\}^n$, let $\mathtt{NaT}[E, H]$ be the nonce-based keyed function with key space $\mathcal{K} \times \mathcal{K}_h$, nonce space $\mathcal{T}$, and domain $\mathcal{D}$ defined as

$$\mathtt{NaT}[E, H]_{K,K_h}(N, D) = E^N_K(H_{K_h}(D)).$$

The following theorem was proved in [19].[10]

Theorem 3. *Let $\mathcal{K}$, $\mathcal{T}$, and $\mathcal{K}_h$ be finite non-empty sets and $\mathcal{D}$ be a non-empty set. Let $E : \mathcal{K} \times \mathcal{T} \times \{0,1\}^n \to \{0,1\}^n$ be a tweakable block cipher and $H : \mathcal{K}_h \times \mathcal{D} \to \{0,1\}^n$ be an ε-AU hash function. Let μ, q, and t be integers such that $q \leq 2^n$ and $\mu \leq \min\{q, 2^n - 1\}$. Then for any adversary A against the nPRM-security of $\mathtt{NaT}[E, H]$ making at most q queries in total to its oracles, repeating any nonce at most μ times in its queries to the first oracle, and running in time at most t, there exists a (q, t')-adversary A' against the* TPRP*-security of E, where $t' = t + qt_H$ and t_H is an upper bound on the time to compute H on any message, such that*

$$\mathbf{Adv}^{\mathtt{nPRM}}_{\mathtt{NaT}[E,H]}(\mathsf{A}) \leq \mathbf{Adv}^{\mathrm{TPRP}}_{E}(\mathsf{A}') + (3\mu - 2)q\varepsilon + \frac{q}{2^n - \mu}.$$

It remains to upper bound the collision probability of the universal hash function used in `SCT-2`. Assume that the tweakable block cipher E is replaced by a uniformly random tweakable permutation $\widetilde{P}$. Let (A, M) be an AD/message pair and write $A = A_1 \| \cdots \| A_{\ell_a} \| A_*$ and $M = M_1 \| \cdots \| M_{\ell_m} \| M_*$, where $|A_i| = |M_j| = n$ and $|A_*| < n$ and $|M_*| < n$. Then, according to Algorithm 5, the value

[10] The statement of the theorem in [19] mentions MAC-security, but the proof actually shows the stronger nPRM notion. Besides, queries to the first and second oracle are counted separately in [19]; we upper bound both of them by q.

tag is computed by encrypting with $\widetilde{P}^{1,0^4\|N}$ the output of the universal hash function (with key $\widetilde{P}$) defined as $H_{\widetilde{P}}(A,M) = H^{\mathrm{ad}}_{\widetilde{P}}(A) \oplus H^{\mathrm{mes}}_{\widetilde{P}}(M)$, where

$$H^{\mathrm{ad}}_{\widetilde{P}}(A) \coloneqq \begin{cases} 0 & \text{if } A = \epsilon \\ \bigoplus_{i=1}^{\ell_{\mathrm{a}}} \widetilde{P}^{2,i-1}(A_i) & \text{if } A \neq \epsilon \text{ and } A_* = \epsilon \\ \left(\bigoplus_{i=1}^{\ell_{\mathrm{a}}} \widetilde{P}^{2,i-1}(A_i)\right) \oplus \widetilde{P}^{6,\ell_{\mathrm{a}}}(\mathsf{ozpad}(A_*)) & \text{if } A_* \neq \epsilon \end{cases}$$

and

$$H^{\mathrm{mes}}_{\widetilde{P}}(M) \coloneqq \begin{cases} 0 & \text{if } M = \epsilon \\ \bigoplus_{i=1}^{\ell_m} \widetilde{P}^{0,i-1}(M_i) & \text{if } M \neq \epsilon \text{ and } M_* = \epsilon \\ \left(\bigoplus_{i=1}^{\ell_m} \widetilde{P}^{0,i-1}(M_i)\right) \oplus \widetilde{P}^{4,\ell_m}(\mathsf{ozpad}(M_*)) & \text{if } M_* \neq \epsilon. \end{cases}$$

We show that for any two distinct inputs (A,M) and (A',M'),

$$H_{\widetilde{P}}(A,M) = H_{\widetilde{P}}(A',M') \tag{4}$$

with probability at most 2^{-n} over the random draw of $\widetilde{P}$. Assume that $A \neq A'$ (the reasoning is similar if $A = A'$ and $M \neq M'$). Let $A = A_1\| \cdots \|A_{\ell_{\mathrm{a}}}\|A_*$ and $A' = A'_1\| \cdots \|A'_{\ell'_{\mathrm{a}}}\|A'_*$ and assume *wlog* that $\ell_{\mathrm{a}} \geq \ell'_{\mathrm{a}}$. We distinguish the following cases:

- $\ell_{\mathrm{a}} > \ell'_{\mathrm{a}}$: then (4) is equivalent to $\widetilde{P}^{2,\ell_{\mathrm{a}}-1}(A_{\ell_{\mathrm{a}}}) = Z$, where Z is independent from $\widetilde{P}^{2,\ell_{\mathrm{a}}-1}$, hence the probability is exactly 2^{-n};
- $\ell_{\mathrm{a}} = \ell'_{\mathrm{a}}$ and $A_* \neq A'_*$: then (4) is equivalent to either $\widetilde{P}^{6,\ell_{\mathrm{a}}}(\mathsf{ozpad}(A_*)) = Z$ (if $A'_* = \epsilon$), $\widetilde{P}^{6,\ell_{\mathrm{a}}}(\mathsf{ozpad}(A'_*)) = Z$ (if $A_* = \epsilon$), or $\widetilde{P}^{6,\ell_{\mathrm{a}}}(\mathsf{ozpad}(A_*)) \oplus \widetilde{P}^{6,\ell_{\mathrm{a}}}(\mathsf{ozpad}(A'_*)) = Z$, where Z is independent from the left hand-side; hence this happens with probability 0 or 2^{-n} depending on whether $Z = 0$ or $Z \neq 0$;
- $\ell_{\mathrm{a}} = \ell'_{\mathrm{a}}$ and $A_* = A'_*$: then there is necessarily an index $i \leq \ell_{\mathrm{a}}$ such that $A_i \neq A'_i$ and (4) is equivalent to $\widetilde{P}^{2,i-1}(A_i) \oplus \widetilde{P}^{2,i-1}(A'_i) = Z$, where Z is independent from $\widetilde{P}^{2,i-1}$; hence this happens with probability 0 or 2^{-n} depending on whether $Z = 0$ or $Z \neq 0$.

From this observation and Theorem 3, we deduce the following result regarding the keyed function used in `SCT-2` defined as

$$F_K(N,A,M) \coloneqq E_K^{1,0^4\|N}(H_{E_K}(A,M)). \tag{5}$$

We let $t_F(\sigma)$ denote an upper bound on the time needed to compute F on inputs of total (AD + message) length at most σ blocks of n bits when calls to E_K cost unit time.

Theorem 4. *Let $E \in \mathsf{TBC}(\mathcal{K},\mathcal{T}',\mathcal{X})$ with $\mathcal{X} = \{0,1\}^n$ and $\mathcal{T}' \coloneqq \{0,1\}^4 \times \mathcal{T}$. Let A be a (q,μ,ℓ,σ,t)-adversary against the nPRM-security of F defined as in*

(5). *Then there exists an adversary* B *against the TPRP-security of* E*, making at most* $\sigma + q$ *oracle queries and running in time at most* $t + t_F(\sigma)$*, such that*

$$\mathbf{Adv}_F^{\mathtt{nPRM}}(\mathsf{A}) \leq \mathbf{Adv}_E^{\mathrm{TPRP}}(\mathsf{B}) + \frac{(3\mu - 2)q}{2^n} + \frac{q}{2^n - \mu}.$$

Proof (sketch). We first replace E_K with a uniformly random tweakable permutation $\widetilde{P}$. Then Theorem 3 applies as tweak separation ensures that the inner universal hash function and the outer tweakable permutation call are independently keyed.

7.5 Security of SCT-2

Observing that $\mathtt{SCT\text{-}2}[E]$ is exactly $\mathtt{NSIV}[F[E], \Pi[E]]$ with $F[E]$ as defined in (5) and $\Pi[E] = \mathtt{CTRT}[E]$, we finally obtain the following result by combining Theorem 1, Theorem 2, and Theorem 4.

Theorem 5. *Let* $E \in \mathsf{TBC}(\mathcal{K}, \mathcal{T}', \mathcal{X})$ *with* $\mathcal{X} = \{0,1\}^n$ *and* $\mathcal{T}' := \{0,1\}^4 \times \{0,1\}^{t-4}$*. Let* A *be a* $(q, \mu, \ell, \sigma, t)$*-adversary against the nAE-security of* $\mathtt{SCT\text{-}2}[E]$*. Then there exists an adversary* B *against the TPRP-security of* E*, making at most* $\sigma + q$ *oracle queries and running in time at most* $t + t_{\mathtt{SCT-2}}(\sigma)$*, such that*

$$\mathbf{Adv}^{\mathrm{nAE}}_{\mathtt{SCT\text{-}2}[E]}(\mathsf{A}) \leq \mathbf{Adv}_E^{\mathrm{TPRP}}(\mathsf{B}) + \frac{2(\mu - 1)\sigma}{2^{t-4}} + \frac{\sigma^2}{2^{n+t-3}} + \frac{(3\mu - 2)q}{2^n} + \frac{q}{2^n - \mu}.$$

Proof (sketch). We first replace E_K with a uniformly random tweakable permutation $\widetilde{P}$ in $\mathtt{SCT\text{-}2}$, then apply Theorem 1 to $\mathtt{NSIV}[F[\widetilde{P}], \mathtt{CTRT}[\widetilde{P}]]$, and finally apply Theorem 2 to $\mathtt{NSIV}[\widetilde{P}]$ and Theorem 4 to $F[\widetilde{P}]$.

8 Software Performances

As Deoxys is based on the AES, it allows very efficient software implementations on the processors that support AES-NI. In addition, the modes allow complete parallelization of the AES-NI calls. The actual overhead compared to AES mostly comes from the increased number of rounds and slightly from the tweak schedule. The key schedule plays role only for very short messages, but even then, it is quite efficient and much faster than the key schedule of AES. Note that the key schedule uses lightweight LFSR to update the subkey bytes, but not the tweak schedule. This was made on purpose, as the subkeys are computed once, while the subtweaks change in each call to the block cipher. For a fixed key, the overhead of the tweak schedule is one XOR and one permutation of 16 bytes, and arguably it is one of the most efficient tweak schedules in the framework of TWEAKEY.

The Deoxys website[11] provides a reference implementation for all Deoxys versions, a table-based implementation, a bitslice implementation, as well as two different AES-NI implementations.

[11] https://sites.google.com/view/deoxyscipher

Our benchmarks are available in Table 9 for nonce-respecting primitives and in Table 10 for nonce-misuse resistant primitives. We measured encryption for several message/associated data size combinations. We used the three Intel processor families Intel Ivy Bridge, Intel Haswell and Intel Skylake, all on Linux with `gcc v9.2`, with `AES-NI` enabled and Turbo Boost disabled to obtain the benchmarks. The reported speed was taken as an average over multiple executions of the code with the same fixed message length. In all measurements, the loading of the bytes from the memory and storing them back to memory are included. The key schedule is also computed for each encryption.

For complete and consistent benchmark comparisons with other authenticated encryption schemes, we refer the interested reader to SUPERCOP available online.[12] For comparison, we also provide benchmarks for `AES-GCM` and `AES-GCM-SIV` respectively with a 128-bit key.

Table 9: Encryption benchmarks for `Deoxys-I-128`, `Deoxys-I-256` and `Deoxys-AE1`, with comparison with `AES-GCM-128`, expressed in cycles per byte on `AES-NI` enabled platforms (with Turbo Boost disabled) for increasing numbers of processed bytes. The key schedule is computed at each call. Benchmarks for `AES-GCM-128` were drawn from SUPERCOP results page (https://bench.cr.yp.to/results-caesar.html).

`Deoxys-I-128` (with key schedule)

AD bytes	0	0	0	64	576	1.5k	64	576	1.5k	0	65k	65k
M bytes	64	576	1.5k	0	0	0	64	576	1.5k	65k	0	65k
Intel Ivy Bridge	3.93	2.13	1.55	5.16	2.36	1.55	3.13	2.12	1.55	1.33	1.31	1.31
Intel Haswell	4.14	1.60	1.19	4.59	1.74	1.19	3.03	1.46	1.11	0.98	0.97	0.97
Intel Skylake	3.06	1.32	1.04	3.38	1.36	1.07	2.05	1.28	0.99	0.89	0.88	0.88

`Deoxys-I-256` (with key schedule)

AD bytes	0	0	0	64	576	1.5k	64	576	1.5k	0	65k	65k
M bytes	64	576	1.5k	0	0	0	64	576	1.5k	65k	0	65k
Intel Ivy Bridge	5.20	2.57	1.82	7.04	2.70	1.86	3.76	2.49	1.83	1.54	1.53	1.53
Intel Haswell	5.60	1.99	1.43	7.05	2.12	1.37	3.91	1.73	1.27	1.10	1.07	1.08
Intel Skylake	4.15	1.65	1.22	4.79	1.67	1.26	2.63	1.49	1.15	1.02	1.01	1.01

`AES-GCM-128` (with key schedule)

AD bytes	0	0	0	64	576	1.5k	64	576	1.5k	0	65k	65k
M bytes	64	576	1.5k	0	0	0	64	576	1.5k	65k	0	65k
Intel Ivy Bridge	-	-	-	-	-	-	12.56	-	2.72	3.23	1.80	2.51
Intel Haswell	-	-	-	-	-	-	13.22	-	1.23	1.05	0.42	0.74
Intel Skylake	-	-	-	-	-	-	10.88	-	0.92	0.66	0.36	0.51

[12] https://bench.cr.yp.to/ebaead.html

Table 10: Encryption benchmarks for `Deoxys-II-128`, `Deoxys-II-256` and `Deoxys-AE2`, with comparison with `AES-GCM-128`, expressed in cycles per byte on `AES-NI` enabled platforms (with Turbo Boost disabled) for increasing numbers of processed bytes. The key schedule is computed at each call. Implementations for `AES-GCM-SIV` were taken from https://github.com/Shay-Gueron/AES-GCM-SIV ("Performance" ones) and benchmarked on our computers.

`Deoxys-II-128` (with key schedule)

AD bytes	0	0	0	64	576	1.5k	64	576	1.5k	0	65k	65k
M bytes	64	576	1.5k	0	0	0	64	576	1.5k	65k	0	65k
Intel Ivy Bridge	8.16	4.16	3.04	6.36	2.31	1.57	5.07	3.11	2.36	2.63	1.32	1.97
Intel Haswell	7.45	2.84	2.17	5.80	1.63	1.13	4.66	2.06	1.58	1.92	0.94	1.43
Intel Skylake	5.80	2.45	1.97	4.56	1.44	1.05	3.45	1.81	1.45	1.77	0.88	1.32

`Deoxys-II-256` (with key schedule)

AD bytes	0	0	0	64	576	1.5k	64	576	1.5k	0	65k	65k
M bytes	64	576	1.5k	0	0	0	64	576	1.5k	65k	0	65k
Intel Ivy Bridge	9.87	4.77	3.45	7.91	2.70	1.79	6.13	3.58	2.74	3.02	1.50	2.27
Intel Haswell	9.55	3.40	2.49	7.69	2.06	1.35	5.83	2.43	1.81	2.18	1.07	1.62
Intel Skylake	7.22	2.92	2.28	6.02	1.75	1.23	4.23	2.12	1.67	2.02	1.01	1.51

`AES-GCM-SIV-128` (with key schedule)

AD bytes	0	0	0	64	576	1.5k	64	576	1.5k	0	65k	65k
M bytes	64	576	1.5k	0	0	0	64	576	1.5k	65k	0	65k
Intel Ivy Bridge	-	-	-	-	-	-	-	-	-	-	-	-
Intel Haswell	10.64	2.56	1.65	9.33	1.83	1.01	6.49	1.59	1.10	1.18	0.55	0.86
Intel Skylake	10.09	2.44	1.55	9.23	1.63	0.82	6.20	1.50	0.95	1.08	0.35	0.71

`AES-GCM-SIV-256` (with key schedule)

AD bytes	0	0	0	64	576	1.5k	64	576	1.5k	0	65k	65k
M bytes	64	576	1.5k	0	0	0	64	576	1.5k	65k	0	65k
Intel Ivy Bridge	-	-	-	-	-	-	-	-	-	-	-	-
Intel Haswell	12.66	3.03	1.97	10.95	2.01	1.08	7.50	1.82	1.26	1.43	0.55	0.99
Intel Skylake	12.96	3.54	1.96	11.47	2.16	1.10	7.17	1.78	1.24	1.36	0.35	0.85

We can see that our nonce-respecting mode `Deoxys-I` performs well, with a throughput similar to `AES-GCM`. We recall that `Deoxys-I` provides beyond-birthday bound security, in contrary to `AES-GCM`. We also observe that our nonce-respecting mode is very efficient for small messages, as no preprocessing step is required to start the cipher calls. The cycle per byte count is already very close to minimal at 1.5k byte size inputs.

Our nonce-misuse resistant mode `Deoxys-II` also performs very well: the efficiency for associated data input being the same as `Deoxys-I`, while the efficiency for message input being about twice slower than `Deoxys-I` (since twice more calls to the internal TBC are required). It is generally slower than

AES-GCM-SIV for very long messages. However, we recall that the Internet Mix is generally composed of mostly very small (smaller than 100-byte) packets, then some medium size (around 500-byte) packets and finally a low proportion of maximum-size (maximum Ethernet packet payload size is 1536 bytes). Deoxys-II actually performs faster than AES-GCM-SIV for small messages, about the same for medium messages and slower for long messages.

Although the above benchmarks rely on the AES-NI instruction set, the simplicity of the Deoxys-TBC tweak schedule guarantees that the speed ratio compared to AES will remain the same even if we used a simple table look-up implementation of AES. In fact, the overhead of the tweak schedule compared to AES will be very small, and the speed of Deoxys-TBC will be very close to the speed of AES. As AES performs rather well on most platforms, we expect the same profile for Deoxys-TBC.

9 Hardware Performances

9.1 ASIC implementations

We first report preliminary ASIC implementations from Axel Poschmann and Marc Stöttinger [56]. *Xlinx ISE DesignSuite 13.3* and *Synopsys DesignCompiler E-2010.12-SP2* were used for functional simulation and synthesis of the designs to the *Virtual Silicon* (VST) standard cell library *UMCL18G212T3* [64], which is based on the *UMC L180 0.18µm 1P6M* logic process with a typical voltage of 1.8 V. For synthesis and for power estimation the compiler was advised to keep the hierarchy and use a clock frequency of 100 KHz.

Different variants of Deoxys in VHDL have been implemented and their post-synthesis performance simulated. Two architectures were designed: one is fully serialized, i.e. performing operations on one cell per clock cycle, and aims for the smallest area possible; the second one is a round-based implementation, thus performing one round in one clock cycle, resulting in a significant speed-up. Only the encryption and authentication parts have been implemented (no decryption capability).

The following tables give hardware performance results independently for the internal tweakable block ciphers and when they are plugged in the modes. Table 11 gives hardware performances of Deoxys-TBC-256 and Deoxys-TBC-384, while Table 12 considers the higher level where the primitives are plugged into the two modes I and II previously described depending on some test cases (Table 13).

Later, in [20], [39] and [45] new ASIC implementations of Deoxys-I and Deoxys-II were reported, all fully CAESAR API compliant, with comparisons with other CAESAR candidates. They were synthesized on TSMC 65nm technology, see Table 14.

9.2 FPGA implementations

We report in Table 15 several third-party FPGA implementations of Deoxys. First, a round-based FPGA implementations from the GMU research team [30],

Table 11: Overview of the ASIC performances of the underlying tweakable block ciphers (UMC L180 0.18μm 1P6M).

Internal Primitive	Architecture	Clk	Area [GE]
`Deoxys-TBC-256`	Round	14	8,005
	Serial	338	2,860
`Deoxys-TBC-384`	Round	16	9,362
	Serial	384	3,575

Table 12: Overview of the ASIC performances for the two modes `I` and `II` for the two tweakable block cipher versions (UMC L180 0.18μm 1P6M). The various test cases definitions are given in Table 13. * denotes slightly modified API.

Mode	TBC	Arch.	Clock Cycles for Test Case							Area [GE]
			I	II	III	IV	V	VI	VII	
`I`	`Deoxys-TBC-256`	**Round**	**22**	**60**	**79**	**62**	**82**	**81**	**120**	**12,496**
		Round*	19	57	76	59	79	78	117	11,936
	`Deoxys-TBC-384`	**Round**	**24**	**66**	**87**	**68**	**90**	**89**	**132**	**13,872**
		Round*	21	63	84	65	87	86	129	13,326
`II`	`Deoxys-TBC-256`	**Round**	**58**	**96**	**115**	**137**	**176**	**156**	**214**	**12,744**
		Round*	55	93	112	134	173	153	211	12,068
	`Deoxys-TBC-384`	**Round**	**64**	**106**	**127**	**151**	**194**	**172**	**236**	**14,107**
		Round*	61	103	124	148	191	169	233	13,422

Table 13: Test cases and length (in bytes) of Associated Data, Message, Key, and Nonce.

Test Case	AD	Message	Key	Nonce
I	0	0	16	8
II	32	0	16	8
III	33	0	16	8
IV	0	32	16	8
V	0	33	16	8
VI	16	32	16	8
VII	17	33	16	8

which are available on the ATHENA website.[13] These implementations are CAESAR Hardware API-compliant implementations (thus with both encryption

[13] https://cryptography.gmu.edu/athena

Table 14: Overview of the ASIC performances of the Deoxys authenticated encryption primitives (TSMC 65nm).

Primitive	Ref.	Area [kGE]	Max Freq. [MHz]	Throughput [Mbps]
Deoxys-I-128	[20]	53.37	549	4,684
	[39]	59.53	847	7,227
	[45]	68.68	532	4,540
	[45]	62.79	448 (min freq.)	3,830
Deoxys-II-128	[45]	53.37	549	2,430
	[45]	45.52	448 (min freq.)	1,980

and decryption capabilities). Then, in [39] and [45] were reported improved FPGA implementations of Deoxys-I, with comparisons with other CAESAR candidates. Again, one can see that Deoxys has naturally the same hardware profile as AES. We refer the reader to the cited papers for more details on the implementation details.

Table 15: Overview of the FPGA performances of the Deoxys authenticated encryption primitives (Virtex 6). Implementations that are not CAESAR API compliant are marked with a †.

Primitive	Ref.	Slices	Max Freq. [MHz]	Throughput [Mbps]
Deoxys-I-128	[20]	956	285	2,432
	[20] †	946	285	2,432
	[39]	805	225	1,920
	[39] †	861	454	3,874
Deoxys-I-128 (encryption only)	[57] †	920	161	1,030
	[39] †	566	416	3,549

Acknowledgements

We would like to thank the anonymous reviewer of the CAESAR committee for their helpful comments and suggestions, Tetsu Iwata, Guo Jian, Gaëtan Leurent, and Wang Lei for very fruitful discussions on authenticated encryption designs, Christof Beierle and Anne Canteaut for pointing out issues in our first general formulations of the bound on the number of active S-boxes coming from the

subtweakeys schedule in the STK construction, and Axel Poschmann and Marc Stöttinger for their hardware implementations of Deoxys presented in Section 9.

This work is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

[1] M. R. Albrecht, K. G. Paterson, and G. J. Watson. Plaintext recovery attacks against SSH. In *2009 IEEE Symposium on Security and Privacy*, pages 16–26. IEEE Computer Society Press, May 2009.

[2] N. J. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540. IEEE Computer Society Press, May 2013.

[3] E. Andreeva, A. Bogdanov, N. Datta, A. Luykx, B. Mennink, M. Nandi, E. Tischhauser, and K. Yasuda. COLM v1. Submission to the CAESAR competition, 2015.

[4] E. Andreeva, A. Bogdanov, A. Luykx, B. Mennink, N. Mouha, and K. Yasuda. How to securely release unverified plaintext in authenticated encryption. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 105–125. Springer, Heidelberg, Dec. 2014.

[5] C. Beierle, J. Jean, S. Kölbl, G. Leander, A. Moradi, T. Peyrin, Y. Sasaki, P. Sasdrich, and S. M. Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 123–153. Springer, Heidelberg, Aug. 2016.

[6] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, Oct. 1997.

[7] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545. Springer, Heidelberg, Dec. 2000.

[8] E. Biham, O. Dunkelman, and N. Keller. The rectangle attack - rectangling the Serpent. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 340–357. Springer, Heidelberg, May 2001.

[9] E. Biham, O. Dunkelman, and N. Keller. New results on boomerang and rectangle attacks. In J. Daemen and V. Rijmen, editors, *FSE 2002*, volume 2365 of *LNCS*, pages 1–16. Springer, Heidelberg, Feb. 2002.

[10] B. Bilgin, A. Bogdanov, M. Knežević, F. Mendel, and Q. Wang. Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In G. Bertoni and J.-S. Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 142–158. Springer, Heidelberg, Aug. 2013.

[11] A. Biryukov and D. Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 1–18. Springer, Heidelberg, Dec. 2009.

[12] A. Biryukov, D. Khovratovich, and I. Nikolic. Distinguisher and related-key attack on the full AES-256. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 231–249. Springer, Heidelberg, Aug. 2009.

[13] A. Biryukov and I. Nikolic. Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to AES, Camellia, Khazad and

others. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 322–344. Springer, Heidelberg, May / June 2010.

[14] A. Biryukov and I. Nikolic. Search for related-key differential characteristics in DES-like ciphers. In A. Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 18–34. Springer, Heidelberg, Feb. 2011.

[15] A. Biryukov and D. Wagner. Slide attacks. In L. R. Knudsen, editor, *FSE'99*, volume 1636 of *LNCS*, pages 245–259. Springer, Heidelberg, Mar. 1999.

[16] A. Bogdanov, F. Mendel, F. Regazzoni, V. Rijmen, and E. Tischhauser. ALE: AES-based lightweight authenticated encryption. In S. Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 447–466. Springer, Heidelberg, Mar. 2014.

[17] C. Cid, T. Huang, T. Peyrin, Y. Sasaki, and L. Song. A security analysis of Deoxys and its internal tweakable block ciphers. *IACR Trans. Symm. Cryptol.*, 2017(3):73–107, 2017.

[18] C. Cid, T. Huang, T. Peyrin, Y. Sasaki, and L. Song. Boomerang connectivity table: A new cryptanalysis tool. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 683–714. Springer, Heidelberg, Apr. / May 2018.

[19] B. Cogliati, J. Lee, and Y. Seurin. New constructions of macs from (tweakable) block ciphers. *IACR Trans. Symm. Cryptol.*, 2017(2):27–58, 2017.

[20] G. M. U. Cryptographic Engineering Research Group. ATHENa: Automated Tools for Hardware EvaluatioN - Deoxys-I-128 implementation, 2016. https://cryptography.gmu.edu/athena/.

[21] H. Demirci and A. A. Selçuk. A meet-in-the-middle attack on 8-round AES. In K. Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 116–126. Springer, Heidelberg, Feb. 2008.

[22] P. Derbez, P.-A. Fouque, and J. Jean. Faster chosen-key distinguishers on reduced-round AES. In S. D. Galbraith and M. Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pages 225–243. Springer, Heidelberg, Dec. 2012.

[23] P. Derbez, P.-A. Fouque, and J. Jean. Improved key recovery attacks on reduced-round AES in the single-key setting. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 371–387. Springer, Heidelberg, May 2013.

[24] I. Dinur and J. Jean. Cryptanalysis of FIDES. In C. Cid and C. Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 224–240. Springer, Heidelberg, Mar. 2015.

[25] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer. Ascon v1.2. Submission to Round 3 of the CAESAR competition, 2016.

[26] O. Dunkelman, N. Keller, and A. Shamir. Improved single-key attacks on 8-round AES-192 and AES-256. In M. Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 158–176. Springer, Heidelberg, Dec. 2010.

[27] S. Emami, S. Ling, I. Nikolic, J. Pieprzyk, and H. Wang. The resistance of PRESENT-80 against related-key differential attacks. *Cryptography and Communications*, 6(3):171–187, 2014.

[28] E. Fleischmann, C. Forler, and S. Lucks. McOE: A family of almost foolproof on-line authenticated encryption schemes. In A. Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 196–215. Springer, Heidelberg, Mar. 2012.

[29] P.-A. Fouque, J. Jean, and T. Peyrin. Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 183–203. Springer, Heidelberg, Aug. 2013.

[30] K. Gaj, J. Kaps, V. Amirineni, M. Rogawski, E. Homsirikamol, and B. Y. Brewster. ATHENa - Automated Tool for Hardware EvaluatioN: Toward Fair and Comprehensive Benchmarking of Cryptographic Hardware Using FPGAs. In *International Conference on Field Programmable Logic and Applications - FPL 2010*, pages 414–421, 2010.

[31] H. Gilbert and T. Peyrin. Super-sbox cryptanalysis: Improved attacks for AES-like permutations. In S. Hong and T. Iwata, editors, *FSE 2010*, volume 6147 of *LNCS*, pages 365–383. Springer, Heidelberg, Feb. 2010.

[32] S. Gueron, A. Langley, and Y. Lindell. AES-GCM-SIV: Specification and Analysis. IACR Cryptology ePrint Archive, Report 2017/168, 2017. Available at http://eprint.iacr.org/2017/168.

[33] V. T. Hoang, T. Krovetz, and P. Rogaway. Robust authenticated-encryption AEZ and the problem that it solves. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 15–44. Springer, Heidelberg, Apr. 2015.

[34] T. Iwata, K. Minematsu, T. Peyrin, and Y. Seurin. ZMAC: A fast tweakable block cipher mode for highly secure message authentication. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 34–65. Springer, Heidelberg, Aug. 2017.

[35] J. Jean, M. Naya-Plasencia, and T. Peyrin. Improved rebound attack on the finalist Grøstl. In A. Canteaut, editor, *FSE 2012*, volume 7549 of *LNCS*, pages 110–126. Springer, Heidelberg, Mar. 2012.

[36] J. Jean, I. Nikolic, and T. Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In P. Sarkar and T. Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 274–288. Springer, Heidelberg, Dec. 2014.

[37] J. Jean, I. Nikolić, T. Peyrin, and Y. Seurin. `Deoxys` v1.41. Submitted to CAESAR, October 2016.

[38] J. Kelsey, T. Kohno, and B. Schneier. Amplified boomerang attacks against reduced-round MARS and Serpent. In B. Schneier, editor, *FSE 2000*, volume 1978 of *LNCS*, pages 75–93. Springer, Heidelberg, Apr. 2001.

[39] M. Khairallah, A. Chattopadhyay, and T. Peyrin. Looting the LUTs: FPGA optimization of AES and AES-like ciphers for authenticated encryption. In A. Patra and N. P. Smart, editors, *INDOCRYPT 2017*, volume 10698 of *LNCS*, pages 282–301. Springer, Heidelberg, Dec. 2017.

[40] D. Khovratovich and I. Nikolic. Rotational cryptanalysis of ARX. In S. Hong and T. Iwata, editors, *FSE 2010*, volume 6147 of *LNCS*, pages 333–346. Springer, Heidelberg, Feb. 2010.

[41] D. Khovratovich and C. Rechberger. The LOCAL attack: Cryptanalysis of the authenticated encryption scheme ALE. In T. Lange, K. Lauter, and P. Lisonek, editors, *SAC 2013*, volume 8282 of *LNCS*, pages 174–184. Springer, Heidelberg, Aug. 2014.

[42] T. Kranz, G. Leander, and F. Wiemer. Linear cryptanalysis: Key schedules and tweakable block ciphers. *IACR Trans. Symm. Cryptol.*, 2017(1):474–505, 2017.

[43] H. Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 310–331. Springer, Heidelberg, Aug. 2001.

[44] T. Krovetz and P. Rogaway. The software performance of authenticated-encryption modes. In A. Joux, editor, *FSE 2011*, volume 6733 of *LNCS*, pages 306–327. Springer, Heidelberg, Feb. 2011.

[45] S. Kumar, J. Haj-Yahya, M. Khairallah, M. A. Elmohr, and A. Chattopadhyay. A comprehensive performance analysis of hardware implementations of CAESAR candidates. Cryptology ePrint Archive, Report 2017/1261, 2017. https://eprint.iacr.org/2017/1261.

[46] R. Li and C. Jin. Meet-in-the-middle attacks on round-reduced tweakable block cipher Deoxys-BC. *IET Information Security*, 13(1):70–75, 2019.

[47] M. Liskov, R. L. Rivest, and D. Wagner. Tweakable block ciphers. *Journal of Cryptology*, 24(3):588–613, July 2011.

[48] D. A. McGrew and J. Viega. The security and performance of the Galois/counter mode (GCM) of operation. In A. Canteaut and K. Viswanathan, editors, *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, Heidelberg, Dec. 2004.

[49] K. Minematsu. Fast decryption: a new feature of misuse-resistant AE. *IACR Trans. Symm. Cryptol.*, 2020(3):87–118, 2020.

[50] F. Moazami, A. Mehrdad, and H. Soleimany. Impossible Differential Cryptanalysis on Deoxys-BC-256. *ISeCure*, 10(2):93–105, 2018.

[51] N. Mouha, Q. Wang, D. Gu, and B. Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In *Information Security and Cryptology - Inscrypt 2011*, pages 57–76, 2011.

[52] C. Namprempre, P. Rogaway, and T. Shrimpton. Reconsidering generic composition. In P. Q. Nguyen and E. Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 257–274. Springer, Heidelberg, May 2014.

[53] I. Nikolic. How to use metaheuristics for design of symmetric-key primitives. In T. Takagi and T. Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 369–391. Springer, Heidelberg, Dec. 2017.

[54] T. Peyrin. Improved differential attacks for ECHO and Grøstl. In T. Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 370–392. Springer, Heidelberg, Aug. 2010.

[55] T. Peyrin and Y. Seurin. Counter-in-tweak: Authenticated encryption modes for tweakable block ciphers. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 33–63. Springer, Heidelberg, Aug. 2016.

[56] A. Poschmann and M. Stöttinger. Personal communication.

[57] A. Poschmann and M. Stottinger. ATHENa: Automated Tools for Hardware EvaluatioN - Deoxys-I-128 implementation, 2016. https://cryptography.gmu.edu/athena/.

[58] P. Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In P. J. Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 16–31. Springer, Heidelberg, Dec. 2004.

[59] P. Rogaway. Nonce-based symmetric encryption. In B. K. Roy and W. Meier, editors, *FSE 2004*, volume 3017 of *LNCS*, pages 348–359. Springer, Heidelberg, Feb. 2004.

[60] P. Rogaway and T. Shrimpton. A provable-security treatment of the key-wrap problem. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 373–390. Springer, Heidelberg, May / June 2006.

[61] Y. Sasaki. Improved related-tweakey boomerang attacks on deoxys-BC. In A. Joux, A. Nitaj, and T. Rachidi, editors, *AFRICACRYPT 18*, volume 10831 of *LNCS*, pages 87–106. Springer, Heidelberg, May 2018.

[62] S. Sun, L. Hu, P. Wang, K. Qiao, X. Ma, and L. Song. Automatic security evaluation and (related-key) differential characteristic search: Application to SIMON, PRESENT, LBlock, DES(L) and other bit-oriented block ciphers. In P. Sarkar

and T. Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 158–178. Springer, Heidelberg, Dec. 2014.
[63] S. Vaudenay. Security flaws induced by CBC padding - applications to SSL, IPSEC, WTLS... In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 534–546. Springer, Heidelberg, Apr. / May 2002.
[64] Virtual Silicon Inc. 0.18 μm VIP Standard Cell Library Tape Out Ready, Part Number: UMCL18G212T3, Process: UMC Logic 0.18 μm Generic II Technology: 0.18μm, July 2004.
[65] D. Wagner. The boomerang attack. In L. R. Knudsen, editor, *FSE'99*, volume 1636 of *LNCS*, pages 156–170. Springer, Heidelberg, Mar. 1999.
[66] H. Wang and T. Peyrin. Boomerang switch in multiple rounds. *IACR Trans. Symm. Cryptol.*, 2019(1):142–169, 2019.
[67] H. Wu. Related-cipher attacks. In R. H. Deng, S. Qing, F. Bao, and J. Zhou, editors, *ICICS 02*, volume 2513 of *LNCS*, pages 447–455. Springer, Heidelberg, Dec. 2002.
[68] H. Wu. ACORN v3. Submission to Round 3 of the CAESAR competition, 2016.
[69] H. Wu. AEGIS v1.1. Submission to Round 3 of the CAESAR competition, 2016.
[70] B. Zhao, X. Dong, and K. Jia. New Related-Tweakey Boomerang and Rectangle Attacks on Deoxys-BC Including BDT Effect. Cryptology ePrint Archive, Report 2020/102, 2020. https://eprint.iacr.org/2020/102.
[71] B. Zhao, X. Dong, K. Jia, and W. Meier. Improved Related-Tweakey Rectangle Attacks on Reduced-round Deoxys-BC-384 and Deoxys-I-256-128. Cryptology ePrint Archive, Report 2020/103, 2020. https://eprint.iacr.org/2020/103.

A AES S-box and constants

A.1 AES S-box and its inverse

We define here the AES S-box $\mathcal{S}$ and its inverse $\mathcal{S}^{-1}$, as an array where the value of $\mathcal{S}(x)$ can be found at the position x in the array.

Table 16: The AES S-box $\mathcal{S}$. The retrieve the value of $\mathcal{S}(x)$, convert x to its hexadecimal representation, and use its four leftmost bits x and four rightmost bits y as coordinates in the table. For example $\mathcal{S}(\texttt{0x25}) = \texttt{0x3F}$.

	y0	y1	y2	y3	y4	y5	y6	y7	y8	y9	yA	yB	yC	yD	yE	yF
0x	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1x	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2x	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3x	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4x	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5x	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6x	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7x	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8x	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9x	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
Ax	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
Bx	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
Cx	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
Dx	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
Ex	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
Fx	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Table 17: The AES inverse S-box $\mathcal{S}^{-1}$. The retrieve the value of $\mathcal{S}(x)$, convert x to its hexadecimal representation, and use its four leftmost bits x and four rightmost bits y as coordinates in the table. For example $\mathcal{S}(\texttt{0x3F}) = \texttt{0x25}$.

	y0	y1	y2	y3	y4	y5	y6	y7	y8	y9	yA	yB	yC	yD	yE	yF
0x	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1x	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2x	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3x	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4x	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5x	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6x	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7x	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8x	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9x	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
Ax	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
Bx	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
Cx	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
Dx	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
Ex	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
Fx	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

B RCON constants

Table 18 below gives the values of constants RCON used in the tweakey scheduling algorithm of the Deoxys.

Table 18: The RCON constants used in the key scheduling algorithm.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
RCON[i]	2f	5e	bc	63	c6	97	35	6a	d4	b3	7d	fa	ef	c5	91	39	72

C Algorithmic Descriptions of the Deoxys Variants

C.1 Deoxys-I

Algorithm 1: The encryption algorithm Deoxys-I.Enc(K, N, A, M).
In the tweak inputs, the value N is encoded on $\log_2(\max_m) = 64$ bits, the integer values i, j, ℓ_a and ℓ_m are encoded on $\log_2(\max_\ell) = 60$ bits.

/* *Associated data* */
$A_1 \| \ldots \| A_{\ell_\mathrm{a}} \| A_* \leftarrow A$ where each $|A_i| = 128$ and $|A_*| < 128$
Auth $\leftarrow 0^{128}$
for $i = 0$ **to** $\ell_\mathrm{a} - 1$ **do**
  Auth $\leftarrow$ Auth $\oplus\, E_K(0010 \| 0^{64} \| i, A_{i+1})$
end
if $A_* \neq \epsilon$ **then**
  Auth $\leftarrow$ Auth $\oplus\, E_K(0110 \| 0^{64} \| \ell_\mathrm{a}, \mathsf{ozpad}(A_*))$
end

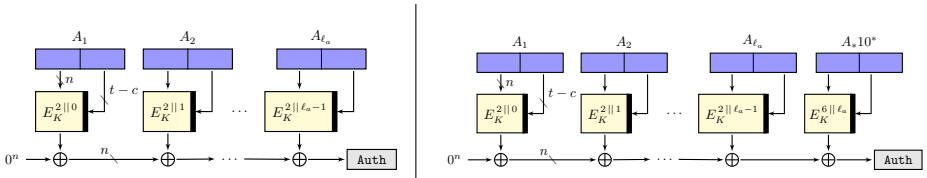
/* *Message* */
$M_1 \| \ldots \| M_{\ell_\mathrm{m}} \| M_* \leftarrow M$ where each $|M_j| = 128$ and $|M_*| < 128$
Checksum $\leftarrow 0^{128}$
for $j = 0$ **to** $\ell_\mathrm{m} - 1$ **do**
  Checksum $\leftarrow$ Checksum $\oplus\, M_{j+1}$
  $C_j \leftarrow E_K(0000 \| N \| j, M_{j+1})$
end
if $M_* = \epsilon$ **then**
  Final $\leftarrow E_K(0001 \| N \| \ell_\mathrm{m}, \text{Checksum})$
  $C_* \leftarrow \epsilon$
else
  Checksum $\leftarrow$ Checksum $\oplus\, \mathsf{ozpad}(M_*)$
  Pad $\leftarrow E_K(0100 \| N \| \ell_\mathrm{m}, 0^{128})$
  $C_* \leftarrow M_* \oplus \lceil \text{Pad} \rceil_{|M_*|}$
  Final $\leftarrow E_K(0101 \| N \| \ell_\mathrm{m} + 1, \text{Checksum})$
end

/* *Tag generation* */
tag $\leftarrow$ Final $\oplus$ Auth
return $(C_1 \| \ldots \| C_{\ell_\mathrm{m}} \| C_*, \text{tag})$

Algorithm 2: The decryption algorithm $\texttt{Deoxys-I.Dec}(K, N, A, C, \text{tag})$. In the tweak inputs, the value N is encoded on $\log_2(\max_m) = 64$ bits, the integer values i, j, ℓ_a and ℓ_m are encoded on $\log_2(\max_\ell) = 60$ bits.

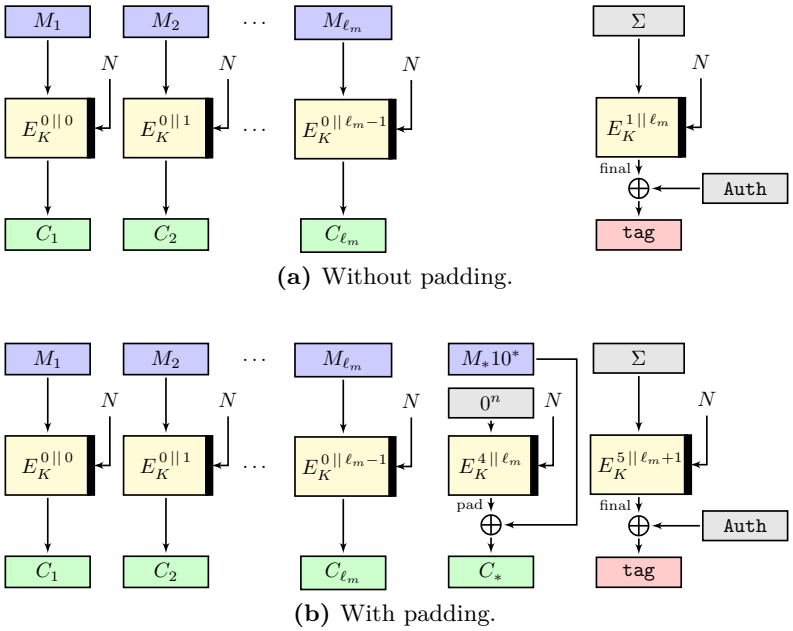
```
/* Associated data */
A_1‖...‖A_{ℓ_a}‖A_* ← A where each |A_i| = 128 and |A_*| < 128
Auth ← 0^128
for i = 0 to ℓ_a − 1 do
    Auth ← Auth ⊕ E_K(0010‖0^64‖i, A_{i+1})
end
if A_* ≠ ϵ then
    Auth ← Auth ⊕ E_K(0110‖0^64‖ℓ_a, ozpad(A_*))
end

/* Ciphertext */
C_1‖...‖C_{ℓ_m}‖C_* ← C where each |C_j| = 128 and |C_*| < 128
Checksum ← 0^128
for j = 0 to ℓ_m − 1 do
    M_{j+1} ← D_K(0000‖N‖j, C_{j+1})
    Checksum ← Checksum ⊕ M_{j+1}
end
if C_* = ϵ then
    Final ← E_K(0001‖N‖ℓ_m, Checksum)
    M_* ← ϵ
else
    Pad ← E_K(0100‖N‖ℓ_m, 0^128)
    M_* ← C_* ⊕ ⌈Pad⌉_{|C_*|}
    Checksum ← Checksum ⊕ ozpad(M_*)
    Final ← E_K(0101‖N‖ℓ_m + 1, Checksum)
end

/* Tag verification */
tag' ← Final ⊕ Auth
if tag' = tag then return (M_1‖...‖M_{ℓ_m}‖M_*)
else return ⊥
```

C.2 Deoxys-AE1



(a) Without padding. **(b)** With padding.

Fig. 10: Handling of the associated data for Deoxys-AE1: in the case where the associated data is a multiple of the block size, no padding is needed.



(a) Without padding.

(b) With padding.

Fig. 11: Message processing for Deoxys-AE1: in the case where the message-length is a multiple of the block size, no padding is needed. Note that the checksum Σ is computed with a 10^* padding for block M^*.

Algorithm 3: The encryption algorithm $\texttt{Deoxys-AE1.Enc}(K, N, A, M)$. In the tweak inputs, the value N is encoded on 128 bits, the integer values i, j, ℓ_{a} and ℓ_{m} are encoded on 120 bits.

```
/* Associated data */
A_1‖...‖A_{ℓa}‖A_* ← A where each |A_i| = 256 and |A_*| < 256
Auth ← 0^128
for i = 0 to ℓa − 1 do
    Auth ← Auth ⊕ E_K(A_{i+1}[0..127]‖00000010‖i, A_{i+1}[128..255])
end
if A_* ≠ ϵ then
    A_pad ← ozpad_256(A_*)
    Auth ← Auth ⊕ E_K(A_pad[0..127]‖00000110‖ℓa, A_pad[128..255])
end

/* Message */
M_1‖...‖M_{ℓm}‖M_* ← M where each |M_j| = 128 and |M_*| < 128
Checksum ← 0^128
for j = 0 to ℓm − 1 do
    Checksum ← Checksum ⊕ M_{j+1}
    C_{j+1} ← E_K(N‖00000000‖j, M_{j+1})
end
if M_* = ϵ then
    Final ← E_K(N‖00000001‖ℓm, Checksum)
    C_* ← ϵ
else
    Checksum ← Checksum ⊕ ozpad_128(M_*)
    Pad ← E_K(N‖00000100‖ℓm, 0^128)
    C_* ← M_* ⊕ ⌈Pad⌉_{|M_*|}
    Final ← E_K(N‖00000101‖ℓm + 1, Checksum)
end

/* Tag generation */
tag ← Final ⊕ Auth
return (C_1‖...‖C_{ℓm}‖C_*, tag)
```

Algorithm 4: The decryption algorithm $\texttt{Deoxys-AE1}.\mathsf{Dec}(K, N, A, C, \text{tag})$.

In the tweak inputs, the value N is encoded on 128 bits, the integer values i, j, ℓ_a and ℓ_m are encoded on 120 bits.

```
/* Associated data */
A_1‖...‖A_{ℓ_a}‖A_* ← A where each |A_i| = 256 and |A_*| < 256
Auth ← 0^128
for i = 0 to ℓ_a − 1 do
    Auth ← Auth ⊕ E_K(A_{i+1}[0..127]‖00000010‖i, A_{i+1}[128..255])
end
if A_* ≠ ϵ then
    A_pad ← ozpad_256(A_*)
    Auth ← Auth ⊕ E_K(A_pad[0..127]‖00000110‖ℓ_a, A_pad[128..255])
end

/* Ciphertext */
C_1‖...‖C_{ℓ_m}‖C_* ← C where each |C_j| = 128 and |C_*| < 128
Checksum ← 0^128
for j = 0 to ℓ_m − 1 do
    M_{j+1} ← D_K(N‖00000000‖j, C_{j+1})
    Checksum ← Checksum ⊕ M_{j+1}
end
if C_* = ϵ then
    Final ← E_K(N‖00000001‖ℓ_m, Checksum)
    M_* ← ϵ
else
    Pad ← E_K(N‖00000100‖ℓ_m, 0^128)
    M_* ← C_* ⊕ ⌈Pad⌉_{|C_*|}
    Checksum ← Checksum ⊕ ozpad_128(M_*)
    Final ← E_K(N‖00000101‖ℓ_m + 1, Checksum)
end

/* Tag verification */
tag′ ← Final ⊕ Auth
if tag′ = tag then return (M_1‖...‖M_{ℓ_m}‖M_*)
else return ⊥
```

C.3 Deoxys-II

Algorithm 5: The encryption algorithm $\texttt{Deoxys-II}.\mathsf{Enc}(K, N, A, M)$.
In the tweak inputs, the integer values i, j, ℓ_{m} and ℓ_{a} are encoded on $\log_2(\max_\ell) = 60$ bits. Recall that the nonce N contains 120 bits.

```
/* Associated data */
A_1‖...‖A_{ℓ_a}‖A_* ← A where each |A_i| = 128 and |A_*| < 128
Auth ← 0^128
for i = 0 to ℓ_a − 1 do
    Auth ← Auth ⊕ E_K(0010‖0^64‖i, A_{i+1})
end
if A_* ≠ ϵ then
    Auth ← Auth ⊕ E_K(0110‖0^64‖ℓ_a, ozpad(A_*))
end

/* Message authentication */
M_1‖...‖M_{lenm}‖M_* ← M where each |M_j| = 128 and |M_*| < 128
for j = 0 to ℓ_m − 1 do
    Auth ← Auth ⊕ E_K(0000‖0^64‖j, M_{j+1})
end
if M_* ≠ ϵ then
    Auth ← Auth ⊕ E_K(0100‖0^64‖ℓ_m, ozpad(M_*))
end

/* Tag generation */
tag ← E_K(0001‖0^4‖N, Auth)

/* Message encryption */
for j = 0 to ℓ_m − 1 do
    C_{j+1} ← M_{j+1} ⊕ E_K((tag ∨ (1‖0^127)) ⊕ (0^68‖j), 0^8 ‖ N)
end
if M_* ≠ ϵ then
    C_* ← M_* ⊕ ⌈E_K((tag ∨ (1‖0^127)) ⊕ (0^68‖ℓ_m), 0^8 ‖ N)⌉_{|M_*|}
end

return (C_1‖...‖C_{ℓ_m}‖C_*, tag)
```

Algorithm 6: The decryption algorithm $\texttt{Deoxys-II}.\mathsf{Dec}(K, N, A, C, \mathrm{tag})$. In the tweak inputs, the integer values i, j, ℓ_m and ℓ_a are encoded on $\log_2(\max_\ell) = 60$ bits. Recall that the nonce N contains 120 bits.

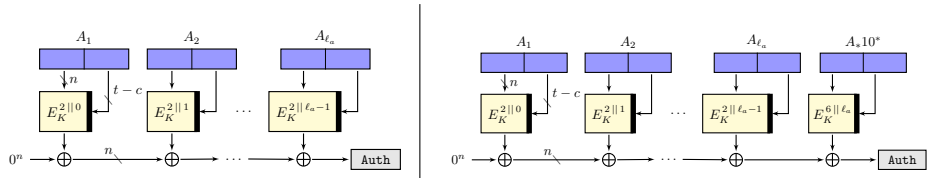
/* *Message decryption* */
$C_1 \| \ldots \| C_{\ell_\mathrm{m}} \| C_* \leftarrow C$ where each $|C_j| = 128$ and $|C_*| < 128$
for $j = 0$ **to** $\ell_\mathrm{m} - 1$ **do**
  $M_{j+1} \leftarrow C_{j+1} \oplus E_K((\mathrm{tag} \vee (1 \| 0^{127})) \oplus (0^{68} \| j), 0^8 \| N)$
end
if $C_* \neq \epsilon$ **then**
  $M_* \leftarrow C_* \oplus \left[E_K((\mathrm{tag} \vee (1 \| 0^{127})) \oplus (0^{68} \| \ell_\mathrm{m}), 0^8 \| N) \right]_{|C_*|}$
end

/* *Associated data* */
$A_1 \| \ldots \| A_{\ell_\mathrm{a}} \| A_* \leftarrow A$ where each $|A_i| = 128$ and $|A_*| < 128$
$\mathrm{Auth} \leftarrow 0^{128}$
for $i = 0$ **to** $\ell_\mathrm{a} - 1$ **do**
  $\mathrm{Auth} \leftarrow \mathrm{Auth} \oplus E_K(0010 \| 0^{64} \| i, A_{i+1})$
end
if $A_* \neq \epsilon$ **then**
  $\mathrm{Auth} \leftarrow \mathrm{Auth} \oplus E_K(0110 \| 0^{64} \| \ell_\mathrm{a}, \mathsf{ozpad}(A_*))$
end

/* *Message authentication* */
for $j = 0$ **to** $\ell_\mathrm{m} - 1$ **do**
  $\mathrm{Auth} \leftarrow \mathrm{Auth} \oplus E_K(0000 \| 0^{64} \| j, M_{j+1})$
end
if $M_* \neq \epsilon$ **then**
  $\mathrm{Auth} \leftarrow \mathrm{Auth} \oplus E_K(0100 \| 0^{64} \| \ell_\mathrm{m}, \mathsf{ozpad}(M_*))$
end

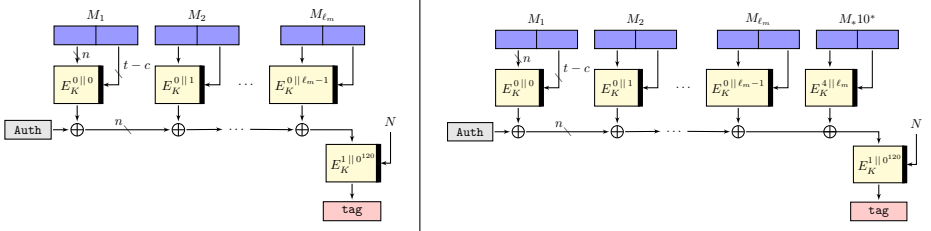
/* *Tag verification* */
$\mathrm{tag}' \leftarrow E_K(0001 \| 0^4 \| N, auth)$
if $\mathrm{tag}' = \mathrm{tag}$ **then return** $(M_1 \| \ldots \| M_{\ell_\mathrm{m}} \| M_*)$
else return $\perp$

C.4 Deoxys-AE2



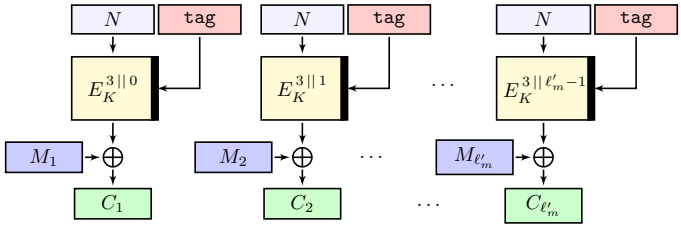
(a) Without padding. **(b)** With padding.

Fig. 12: Handling of the associated data for Deoxys-AE2: in the case where the associated data is a multiple of the block size, no padding is needed.

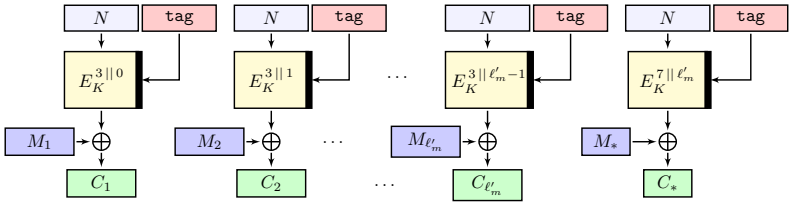


(a) Without padding. **(b)** With padding.

Fig. 13: Message processing in the authentication part of Deoxys-AE2: in the case where the message-length is a multiple of the block size, no padding is needed.



(a) Message-length is a multiple of the block size.



(b) Message-length is not a multiple of the block size.

Fig. 14: Message processing for the encryption part of `Deoxys-AE2`.

Algorithm 7: The encryption algorithm $\texttt{Deoxys-AE2}.\mathsf{Enc}(K, N, A, M)$. In the tweak inputs, the value N is encoded on 128 bits, the integer values i, j, $\ell_{\rm a}$, $\ell_{\rm m}$ and $\ell'_{\rm m}$ are encoded on 120 bits.

/* *Associated data* */
$A_1 \| \ldots \| A_{\ell_{\rm a}} \| A_* \leftarrow A$ where each $|A_i| = 256$ and $|A_*| < 256$
Auth $\leftarrow 0^{128}$
for $i = 0$ **to** $\ell_{\rm a} - 1$ **do**
  Auth $\leftarrow$ Auth $\oplus\, E_K(A_{i+1}[0..127] \| 00000010 \| i, A_{i+1}[128..255])$
end
if $A_* \neq \epsilon$ **then**
  $A_{pad} \leftarrow \mathsf{ozpad}_{256}(A_*)$
  Auth $\leftarrow$ Auth $\oplus\, E_K(A_{pad}[0..127] \| 00000110 \| \ell_{\rm a}, A_{pad}[128..255])$
end

/* *Message authentication* */
$M_1 \| \ldots \| M_{\ell_{\rm m}} \| M_* \leftarrow M$ where each $|M_j| = 256$ and $|M_*| < 256$
for $j = 0$ **to** $\ell_{\rm m} - 1$ **do**
  Auth $\leftarrow$ Auth $\oplus\, E_K(M_{j+1}[0..127] \| 00000000 \| j, M_{j+1}[128..255])$
end
if $M_* \neq \epsilon$ **then**
  $M_{pad} \leftarrow \mathsf{ozpad}_{256}(M_*)$
  Auth $\leftarrow$ Auth $\oplus\, E_K(M_{pad}[0..127] \| 00000100 \| \ell_{\rm m}, M_{pad}[128..255])$
end

/* *Tag generation* */
tag $\leftarrow E_K(N \| 00000001 \| 0^{120}, \text{Auth})$

/* *Message encryption* */
$M_1 \| \ldots \| M_{\ell'_{\rm m}} \| M_* \leftarrow M$ where each $|M_j| = 128$ and $|M_*| < 128$
for $j = 0$ **to** $\ell'_{\rm m} - 1$ **do**
  $C_{j+1} \leftarrow M_{j+1} \oplus E_K(\text{tag} \| 00000011 \| j, N)$
end
if $M_* \neq \epsilon$ **then**
  $C_* \leftarrow M_* \oplus \lceil E_K(\text{tag} \| 00000111 \| \ell'_{\rm m}, N) \rceil_{|M_*|}$
end

return $(C_1 \| \ldots \| C_{\ell'_{\rm m}} \| C_*, \text{tag})$

Algorithm 8: The decryption algorithm $\texttt{Deoxys-AE2}.\mathsf{Dec}(K, N, A, C, \mathrm{tag})$. In the tweak inputs, the value N is encoded on 128 bits, the integer values i, j, ℓ_a, ℓ_m and ℓ'_m are encoded on 120 bits.

/* *Message decryption* */
$C_1 \| \ldots \| C_{\ell'_\mathrm{m}} \| C_* \leftarrow C$ where each $|C_j| = 128$ and $|C_*| < 128$
for $j = 0$ **to** $\ell'_\mathrm{m} - 1$ **do**
  $M_{j+1} \leftarrow C_{j+1} \oplus E_K(\mathrm{tag} \| 00000011 \| j, N)$
end
if $C_* \neq \epsilon$ **then**
  $M_* \leftarrow C_* \oplus \lceil E_K(\mathrm{tag} \| 00000111 \| \ell'_\mathrm{m}, N) \rceil_{|C_*|}$
end

/* *Associated data* */
$A_1 \| \ldots \| A_{\ell_\mathrm{a}} \| A_* \leftarrow A$ where each $|A_i| = 256$ and $|A_*| < 256$
$\mathrm{Auth} \leftarrow 0^{128}$
for $i = 0$ **to** $\ell_\mathrm{a} - 1$ **do**
  $\mathrm{Auth} \leftarrow \mathrm{Auth} \oplus E_K(A_{i+1}[0..127] \| 00000010 \| i, A_{i+1}[128..255])$
end
if $A_* \neq \epsilon$ **then**
  $A_{pad} \leftarrow \mathsf{ozpad}_{256}(A_*)$
  $\mathrm{Auth} \leftarrow \mathrm{Auth} \oplus E_K(A_{pad}[0..127] \| 00000110 \| \ell_\mathrm{a}, A_{pad}[128..255])$
end

/* *Message authentication* */
$M \leftarrow M_1 || \ldots || M_{\ell'_\mathrm{m}} || M_*$
$M_1 \| \ldots \| M_{\ell_\mathrm{m}} \| M_* \leftarrow M$ where each $|M_j| = 256$ and $|M_*| < 256$
for $j = 0$ **to** $\ell_\mathrm{m} - 1$ **do**
  $\mathrm{Auth} \leftarrow \mathrm{Auth} \oplus E_K(M_{j+1}[0..127] \| 00000000 \| j, M_{j+1}[128..255])$
end
if $M_* \neq \epsilon$ **then**
  $M_{pad} \leftarrow \mathsf{ozpad}_{256}(M_*)$
  $\mathrm{Auth} \leftarrow \mathrm{Auth} \oplus E_K(M_{pad}[0..127] \| 00000100 \| \ell_\mathrm{m}, M_{pad}[128..255])$
end

/* *Tag verification* */
$\mathrm{tag}' \leftarrow E_K(N \| 00000001 \| 0^{120}, \mathrm{Auth})$
if $\mathrm{tag}' = \mathrm{tag}$ **then return** M
else return $\perp$