Deoxys v1

Designers/Submitters:

Jérémy Jean, Ivica Nikolić, Thomas Peyrin

Division of Mathematical Sciences,
School of Physical and Mathematical Science,
Nanyang Technological University, Singapore

{JJean,INikolic,Thomas.Peyrin}@ntu.edu.sg

March 16, 2014

Chapter 1

Introduction

In this note, we propose Deoxys, a new authenticated encryption design based on a tweakable block cipher Deoxys-BC using the well-studied AES round function as a building block. We suggest several sets of parameters that can use different key and tweak sizes, and claim security levels for all the parameters in later sections. Our design uses a particular instantiation of a more general framework (so-called TWEAKEY [18]) allowing designers to unify the vision of key and tweak inputs of a cipher. We plug this cipher into two different authenticated encryption modes inspired from fully parallel and provably secure modes OCB [20] and COPA [1] respectively.

In short, Deoxys is an authenticated encryption scheme that provides full 128-bit security (in contrary to AES-GCM [22] or OCB [20]) for both privacy and authenticity. It performs well in software, being faster than AES-GCM [22] on most processors. Moreover, Deoxys performs particularly well for small messages (only $m + 1$ block cipher calls are required for a m block message and no precomputation is required). Switching to a birthday-bound security nonce-misuse resistant variant of Deoxys is made very simple as a tweakable block cipher is a very handy primitive to build an authenticated encryption scheme. Finally, Deoxys can be lightweight (using existing AES lightweight implementation, the extra area mainly consisting in 192 extra bits of memory for the mode and to store the tweak) and the key can be hardcoded for further smaller area footprint.

Organization of the paper. In Chapter 2, we provide the specification of our proposal Deoxys, including the description of the TWEAKEY framework and the sets of parameters for this proposal. In Chapter 3, we precise the security claims for different scenarios for the various parameters, and in Chapter 4 we perform some security analysis regarding this proposal. In Chapters 5 and 6, we detail some design decisions, and finish with Chapters 9 and 10 where we give notes on intellectual property and consent.

Chapter 2

Specification

In this chapter, we present a full specification of our proposal `Deoxys`. We first give the recommended parameter sets and then proceed with the description of the design. We explain the two authenticated encryption modes $\texttt{Deoxys}^{\neq}$ and $\texttt{Deoxys}^{=}$, and then we describe the ad-hoc `AES`-based tweakable block cipher `Deoxys-BC` (which is based on the `TWEAKEY` framework [18]) used to instantiate the modes.

We first introduce some notations. We denote $E_K(T, P)$ the ciphering of the n-bit plaintext P with the tweakable block cipher `Deoxys-BC` with k-bit key K and t-bit tweak T (similarly, D represents the deciphering process). The concatenation operation is represented by $||$ and $pad10^*$ is the function that applies the 10^* padding on n bits, i.e. $pad10^*(X) = X||1||0^{n-|X|-1}$ when $|X| < n$. In contrary, $unpad10^*$ is the function that removes the 10^* padding on n bits, i.e. $unpad10^*(X)$ removes all the consecutive 0 bits in X starting from the right (possibly none, possibly all). The truncation of the word X to the first i bits is given by $trunc_i(X)$. Finally, ϵ will represent the empty string.

Our authenticated encryption scheme `Deoxys` is composed of an encryption part and a verification/decryption part. The encryption part $\mathcal{E}$ takes as input a variable-length plaintext M (with $m = |M|$), a variable-length associated data A (with $a = |A|$), a fixed-length public message number N and a k-bit key K (we deliberately used the same letter K to represent the key in the authenticated encryption scheme and the one in the tweakable block cipher, since they always refer to the same object). It outputs a m-bit ciphertext C and a τ-bit tag $\texttt{tag}$ (with $\tau \in [0, \ldots, n]$), i.e. $(C, \texttt{tag}) = \mathcal{E}_K(N, A, M)$. The verification/decryption part $\mathcal{D}$ takes as input a variable-length ciphertext C (with $m = |C|$), a τ-bit tag $\texttt{tag}$ (with $\tau \in [0, \ldots, n]$), a variable-length associated data A (with $a = |A|$), a fixed-length public message number N and a k-bit key K. It outputs either an error string $\perp$ to signify that the verification failed, or a m-bit string $M = \mathcal{D}_K(N, A, C, \texttt{tag})$ when the tag is valid. The maximum message length (in n-bit blocks) is denoted max_l and the maximum number of messages that can be handled with the same key is denoted max_m. We have that $max_l = 2^{\lceil t/2 \rceil - 3}$ and $max_m = 2^{\lfloor t/2 \rfloor}$. This will ensure that as long as different fixed-length public message numbers (i.e. nonces) are used, the tweak inputs of all the tweakable block cipher calls are all unique. This also naturally implies that $|N| = \log_2(max_m) = \lfloor t/2 \rfloor$. Note that there is a tradeoff possible here between max_l and max_m, as long as $max_l \cdot \mathrm{max}_m = 2^{t-3}$.

2.1 Parameters

`Deoxys` has one parameter: key length k which is a value between 128 bits and 256 bits. The tag size τ is fixed to 128 bits, while the public message length $|N|$ is fixed to 64 bits. The key and the public message lengths are used in the ad-hoc tweakable cipher we define later. We instantiate two modes with this cipher: the first is for nonce-respecting adversaries (denoted with a $\neq$ sign), while the second for nonce misuse-resistant adversaries (denoted with a $=$ sign). For this reason, we introduce another parameter, that signals the mode of our authenticated encryption scheme.

2.2 Recommended Parameter Sets

For each of the two modes we recommend two parameter sets (hence in total we have 4 sets), listed in Table 2.1. For a specific mode, we do not have preferred parameters set, however our preferred mode is the nonce-respecting mode (denoted as $\texttt{Deoxys}^{\neq}$). We denote by $\texttt{Deoxys}^{=}$ the design in the case of the nonce-reusing mode.

Name	**k**	**t**	**n**	$\lvert N\rvert$	τ
$\texttt{Deoxys}^{\neq}$`-128-128`	128	128	128	64	128
$\texttt{Deoxys}^{\neq}$`-256-128`	256	128	128	64	128
$\texttt{Deoxys}^{=}$`-128-128`	128	128	128	64	128
$\texttt{Deoxys}^{=}$`-256-128`	256	128	128	64	128

Table 2.1: Recommended parameter sets for `Deoxys`. Parameters k, t, $|N|$ and τ are related to the signature of the inner tweakable block cipher of `Deoxys`.

2.3 Authenticated Encryption

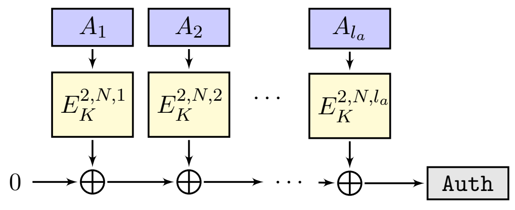
In this section, we provide the high-level description of our proposal. `Deoxys` uses a tweakable block cipher `Deoxys-BC` as internal primitive (specified in Section 2.4), and we describe here the simple authenticated encryption modes built on top of it. `Deoxys` has two main mode variants:

- $\mathcal{E}^{\neq}$ and $\mathcal{D}^{\neq}$ (see Section 2.3.1): the first variant is for where adversaries are assumed to be nonce-respecting, meaning that the user must ensure that the value N will never be used for encryption twice with the same key. This mode is largely inspired from Θ`CB3` [20], the tweakable block cipher generalization of `OCB3`. We will denote $\mathcal{E}^{\neq}$ the encryption part of this first variant (and $\mathcal{D}^{\neq}$ the verification/decryption part).

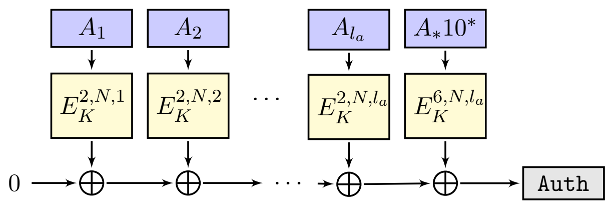
- $\mathcal{E}^{=}$ and $\mathcal{D}^{=}$ (see Section 2.3.2): the second variant, quite close to the first one and inspired by `COPA` mode [1], relaxes this constraint and allows the user to reuse the same N with the same key. We will denote $\mathcal{E}^{=}$ the encryption part of this first variant (and $\mathcal{D}^{=}$ the verification/decryption part).

2.3.1 Nonce-Respecting Mode: $\mathcal{E}^{\neq}$ and $\mathcal{D}^{\neq}$

The encryption algorithm $\mathcal{E}^{\neq}$ is depicted in Figures 2.1, 2.2 and 2.3, and an algorithmic description is given in Algorithm 1. The verification/decryption algorithmic description of $\mathcal{D}^{\neq}$ is given in Algorithm 2. We note that our scheme follows the framework from Θ`CB3` [20] and therefore directly benefits from the security proof regarding authentication and privacy.



(a) Without padding.



(b) With padding.

Figure 2.1: Handling of the associated data for the nonce-respecting mode.

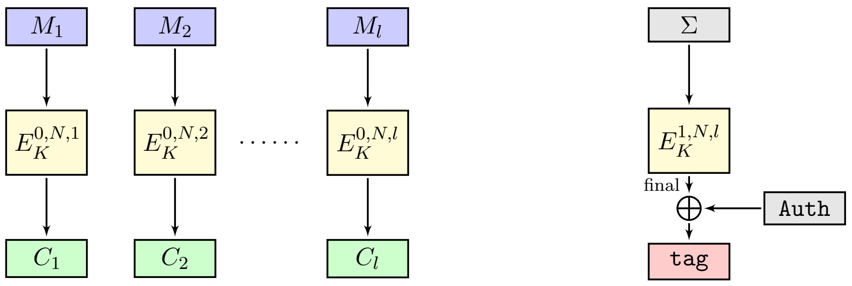


Figure 2.2: Message processing: in the case where the message-length is a multiple of the block size: no padding needed.

Algorithm 1: The encryption algorithm $\mathcal{E}_K^{\neq}(N, A, M)$.
The value N is encoded on $\log_2(max_m)$ bits, while the integer values i, l and l_a are encoded on $\log_2(max_l)$ bits.

```
/* Associated data */
$A_1||\dots||A_{l_a}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$
Auth $\leftarrow 0^n$
for $i = 1$ to $l_a$ do
    Auth $\leftarrow$ Auth $\oplus E_K(010||N||i, A_i)$
end
if $A_* \neq \epsilon$ then
    Auth $\leftarrow$ Auth $\oplus E_K(110||N||l_a, pad10^*(A_*))$
end

/* Message */
$M_1||\dots||M_l||M_* \leftarrow M$ where each $|M_i| = n$ and $|M_*| < n$
Checksum $\leftarrow 0^n$
for $i = 1$ to $l$ do
    Checksum $\leftarrow$ Checksum $\oplus M_i$
    $C_i \leftarrow E_K(000||N||i, M_i)$
end
if $M_* = \epsilon$ then
    Final $\leftarrow E_K(001||N||l, \text{Checksum})$
    $C_* \leftarrow \epsilon$
else
    Checksum $\leftarrow$ Checksum $\oplus pad10^*(M_*)$
    Pad $\leftarrow E_K(100||N||l, 0^n)$
    $C_* \leftarrow M_* \oplus trunc_{|M_*|}(\text{Pad})$
    Final $\leftarrow E_K(101||N||l, \text{Checksum})$
end

/* Tag generation */
tag $\leftarrow trunc_\tau(\text{Final} \oplus \text{Auth})$

return $(C_1||\dots||C_l||C*, \texttt{tag})$
```

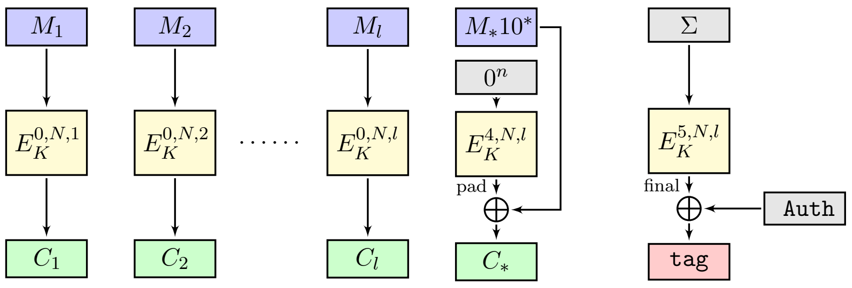


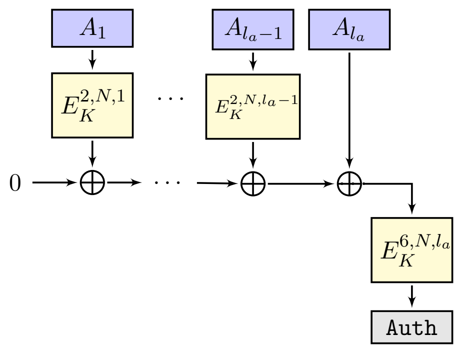
Figure 2.3: Message processing: in the case where the message-length is a not multiple of the block size. Note that the checksum Σ is computed with a 10^* padding for block M^*.

Algorithm 2: The verification/decryption algorithm $\mathcal{D}_K^{\neq}(N, A, C, \texttt{tag})$.
The value N is encoded on $\log_2(max_m)$ bits, while the integer values i, l and l_a are encoded on $\log_2(max_l)$ bits.

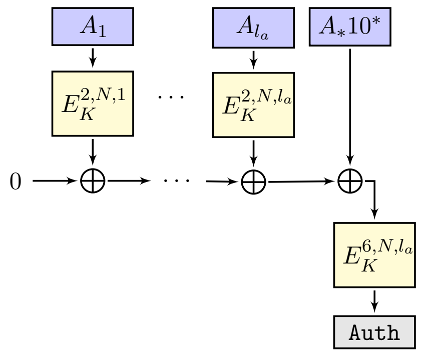
```
/* Associated data */
$A_1||\dots||A_{l_a}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$
Auth $\leftarrow 0^n$
for $i = 1$ to $l_a$ do
    Auth $\leftarrow$ Auth $\oplus E_K(010||N||i, A_i)$
end
if $A_* \neq \epsilon$ then
    Auth $\leftarrow$ Auth $\oplus E_K(110||N||l_a, pad10^*(A_*))$
end

/* Ciphertext */
$C_1||\dots||C_l||C_* \leftarrow C$ where each $|C_i| = n$ and $|C_*| < n$
Checksum $\leftarrow 0^n$
for $i = 1$ to $l$ do
    $M_i \leftarrow D_K(000||N||i, C_i)$
    Checksum $\leftarrow$ Checksum $\oplus M_i$
end
if $C_* = \epsilon$ then
    Final $\leftarrow E_K(001||N||l, \text{Checksum})$
    $M_* \leftarrow \epsilon$
else
    Pad $\leftarrow E_K(100||N||l, 0^n)$
    $M_* \leftarrow C_* \oplus trunc_{|C_*|}(\text{Pad})$
    Checksum $\leftarrow$ Checksum $\oplus pad10^*(M_*)$
    Final $\leftarrow E_K(101||N||l, \text{Checksum})$
end

/* Tag verification */
tag' $\leftarrow trunc_\tau(\text{Final} \oplus \text{Auth})$
if tag' = tag then
    return $(M_1||\dots||M_l||M*)$
else
    return $\perp$
end
```

2.3.2 Nonce-Misuse Resistant Mode: $\mathcal{E}^=$ and $\mathcal{D}^=$

The encryption algorithm $\mathcal{E}^=$ is depicted in Figures 2.4, 2.5 and 2.6 and an algorithmic description is given in Algorithm 3. The verification/decryption algorithmic description of $\mathcal{D}^=$ is given in Algorithm 4. We note that our scheme is a direct adaptation from the COPA [1] construction. If the message is not a multiple of n bits, it goes through a 10^* padding and the ciphertext size might be bigger than the message size. We kept this variant in order to ease the description of our scheme. However, a solution which overcomes this issue is provided in the COPA article [1], where tag splitting is used for messages with $|M| < n$ and XLS [26] for messages with $|M| > n$.



(a) Without padding.



(b) With padding.

Figure 2.4: Handling of the associated data for the nonce-respecting mode.

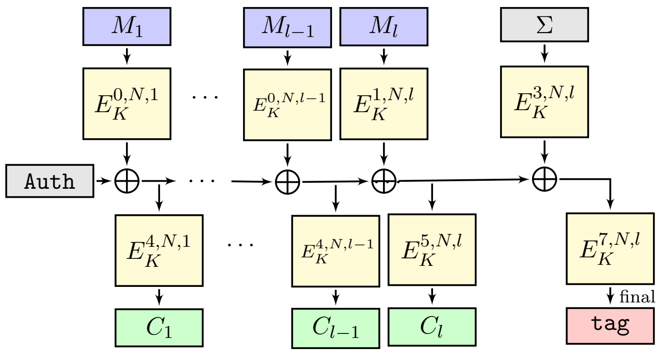


Figure 2.5: Message processing: in the case where the message-length is a multiple of the block size: no padding needed.

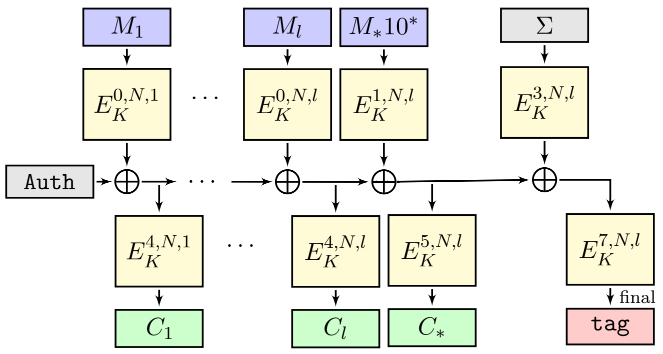


Figure 2.6: Message processing: in the case where the message-length is a not multiple of the block size. Note that the checksum Σ is computed with a 10^* padding for block M^*.

Algorithm 3: The encryption algorithm $\mathcal{E}_K^{=}(N, A, M)$.
The value N is encoded on $\log_2(max_m)$ bits, while the integer values i, l and l_a are encoded on $\log_2(max_l)$ bits.

```
/* Associated data */
$A_1 || \dots || A_{l_a} || A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$
$\text{Auth} \leftarrow 0^n$
for $i = 1$ to $l_a - 1$ do
    $\text{Auth} \leftarrow \text{Auth} \oplus E_K(010||N||i, A_i)$
end
if $A_* \neq \epsilon$ then
    $\text{Auth} \leftarrow \text{Auth} \oplus E_K(010||N||l_a, A_{l_a})$
    $\text{Auth} \leftarrow \text{Auth} \oplus pad10^*(A_*)$
else
    $\text{Auth} \leftarrow \text{Auth} \oplus A_{l_a}$
end
$\text{Auth} \leftarrow E_K(110||N||l_a, \text{Auth})$

/* Message */
$M_1 || \dots || M_l || M_* \leftarrow M$ where each $|M_i| = n$ and $|M_*| < n$
$\text{Checksum} \leftarrow 0^n$
for $i = 1$ to $l - 1$ do
    $\text{Checksum} \leftarrow \text{Checksum} \oplus M_i$
    $\text{Auth} \leftarrow \text{Auth} \oplus E_K(000||N||i, M_i)$
    $C_i \leftarrow E_K(100||N||i, \text{Auth})$
end
if $M_* \neq \epsilon$ then
    $\text{Checksum} \leftarrow \text{Checksum} \oplus M_l$
    $\text{Auth} \leftarrow \text{Auth} \oplus E_K(000||N||l, M_l)$
    $C_l \leftarrow E_K(100||N||l, \text{Auth})$
    $M_* \leftarrow pad10^*(M_*)$
    $\text{Checksum} \leftarrow \text{Checksum} \oplus M_*$
    $\text{Auth} \leftarrow \text{Auth} \oplus E_K(001||N||l, M_*)$
    $C_* \leftarrow E_K(101||N||l, \text{Auth})$
else
    $\text{Checksum} \leftarrow \text{Checksum} \oplus M_l$
    $\text{Auth} \leftarrow \text{Auth} \oplus E_K(001||N||l, M_l)$
    $C_l \leftarrow E_K(101||N||l, \text{Auth})$
    $C_* \leftarrow \epsilon$
end

/* Tag generation */
$\text{Auth} \leftarrow \text{Auth} \oplus E_K(011||N||l, \text{Checksum})$
$\text{Final} \leftarrow E_K(111||N||l, \text{Auth})$
$\texttt{tag} \leftarrow trunc_\tau(\text{Final})$

return $(C_1 || \dots || C_l || C*, \texttt{tag})$
```

Algorithm 4: The verification/decryption algorithm $\mathcal{D}^{=}_{K}(N, A, C, \mathtt{tag})$.
The value N is encoded on $\log_2(max_m)$ bits, while the integer values i, l and l_a are encoded on $\log_2(max_l)$ bits.

```
/* Associated data */
$A_1||\ldots||A_{l_a}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$
Auth $\leftarrow 0^n$
for $i = 1$ to $l_a - 1$ do
    Auth $\leftarrow$ Auth $\oplus E_K(010||N||i, A_i)$
end
if $A_* \neq \epsilon$ then
    Auth $\leftarrow$ Auth $\oplus E_K(010||N||l_a, A_{l_a})$
    Auth $\leftarrow$ Auth $\oplus pad10^*(A_*)$
else
    Auth $\leftarrow$ Auth $\oplus A_{l_a}$
end
Auth $\leftarrow E_K(110||N||l_a, \text{Auth})$

/* Ciphertext */
$C_1||\ldots||C_l \leftarrow C$ where each $|C_i| = n$
Checksum $\leftarrow 0^n$
for $i = 1$ to $l - 1$ do
    $X_i \leftarrow D_K(100||N||i, C_i)$
    $M_i \leftarrow D_K(000||N||i, \text{Auth} \oplus X_i)$
    Checksum $\leftarrow$ Checksum $\oplus M_i$
    Auth $\leftarrow X_i$
end
$X_l \leftarrow D_K(101||N||l, C_l)$
$M_l \leftarrow D_K(001||N||l, \text{Auth} \oplus X_l)$
Checksum $\leftarrow$ Checksum $\oplus M_l$
Auth $\leftarrow X_l$
$M_l \leftarrow unpad10^*(M_l)$

/* Tag verification */
Auth $\leftarrow$ Auth $\oplus E_K(011||N||l, \text{Checksum})$
Final $\leftarrow E_K(111||N||l, \text{Auth})$
$\mathtt{tag}' \leftarrow trunc_\tau(\text{Final})$
if $\mathtt{tag}' = \mathtt{tag}$ then
    return $(M_1||\ldots||M_l)$
else
    return $\perp$
end
```

2.4 The Tweakable Block Cipher Deoxys-BC

Deoxys-BC is an ad-hoc tweakable block cipher so that besides the two standard inputs, a plaintext P (or a ciphertext C) and a key K, it takes an additional input called a tweak T. The cipher $E_K(T, P)$ has 128-bit state and variable size key and tweak. The encryption and decryption are defined in a standard way for tweakable ciphers, i.e. $E_K(T, P) = C$ and $E_K^{-1}(T, C) = P$. We define two ciphers, Deoxys-BC-128 for which the cumulative size of the key and the tweak is 256 bits, and Deoxys-BC-384 for which the cumulative size of the key and the tweak is 384 bits.

Deoxys-BC is an AES-like design, i.e. it is an iterative substitution-permutation network (SPN) that transforms the initial plaintext through series of round functions (that depend on the key and the tweak) to a ciphertext. As most AES-like designs, the state of Deoxys-BC is seen as 4×4 matrix of bytes (we denote c the size of a cell, i.e. $c = 8$). We denote $\mathbb{K}$ the base field as $GF(256)$ defined by the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. The number r of rounds is 14 for Deoxys-BC-128-128 and 16 for Deoxys-BC-256-128. One round, similarly to a round in AES, has 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,
- SubNibbles – Apply the 4-bit S-Box AES $\mathcal{S}$ to the 16 nibbles of the internal state (see Definition in Appendix A.1),
- ShiftRows – Rotate the 4-nibble i-th row left by $\rho[i]$ positions, where $\rho = (0, 1, 2, 3)$.
- MixNibbles – Multiply the internal state by the 4×4 constant MDS matrix $\mathbf{M} \in \mathbb{K}$ defined below.

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

The MDS matrix $\mathbf{M}$ we use in the one from the AES:

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

The round function f^{-1} for a decryption round, naturally, is similar as for the encryption, and the inverse of the four round permutations are applied in a reversed order. We also note that the subtweakeys are used in reverse order. Namely, we perform r times the following operations:

- InvAddRoundTweakey – XOR the 128-bit round subtweakey to the internal state,
- invMixNibbles – Multiply the internal state by the 4×4 MDS matrix $\mathbf{M}^{-1} \in \mathbb{K}$,
- InvShiftRows – Rotate the 4-nibble i-th row *right* by $\rho[i]$ positions, where $\rho = (0, 1, 2, 3)$,
- InvSubNibbles – Apply the inverse 4-bit S-Box S^{-1} to the 16 nibbles of the internal state (see Appendix A.1 for actual values).

Finally, a final InvAddRoundTweakey operation is performed to produced the plaintext value. For the sake of completeness, the inverse of the $\mathbf{M}$ matrix is applied in the inverse of the MixNibbles operations, and its value equals:

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

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-BC differs from the other ciphers.

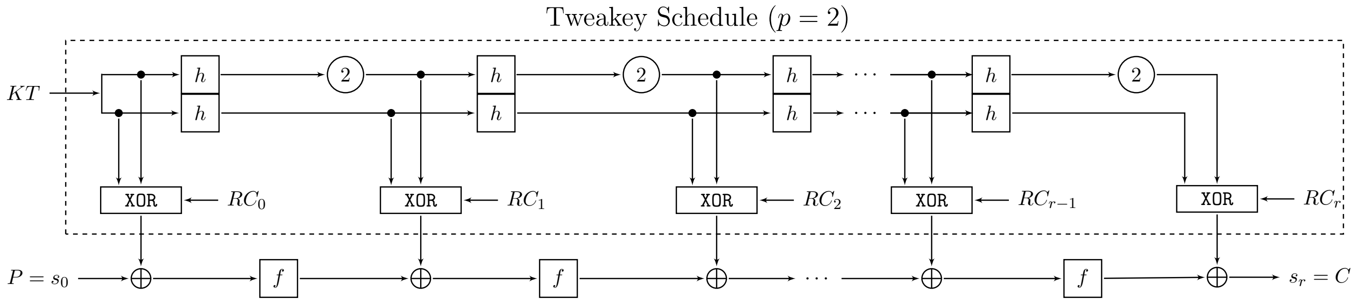


Figure 2.7: Instantiation of the TWEAKEY framework for Deoxys-BC.

Let us denote with STK_i the subtweakey (a 128-bit word) that is added to the state at round i of the cipher with the AddRoundTweakey operation. For Deoxys-BC-128-128, a subtweakey is defined as:

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

whereas for the case of size 384 bits (Deoxys-BC-256-128) 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 special key schedule algorithm. A single instance of this algorithm, denoted as $KS(W, \alpha)$, takes as inputs a 128-bit word W and a nibble α and (as any other key schedule) produces subkeys $TK_0, TK_1, \ldots$. The subkeys are produced sequentially, one from another (where $TK_0 = W$), by applying two permutations: a byte permutation h, and a finite field multiplication g:

$$TK_{i+1} = g(h(TK_i)).$$

The byte permutation h is defined as:

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

where we number the 16 nibbles of the internal state by the usual ordering:

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

Furthermore, g is finite field multiplication of each nibble by α (recall that α is input to the key schedule algorithm).

Let us define the inputs W and α. Denote the concatenation of the key K and the tweak T as KT, i.e. $KT = K||T$. Then, in Deoxys-BC-128-128, the size of KT is 256 bits. The first (most significant) 128 bits of KT is W_1, while the second W_2. Then, TK_1^i are the output words of the key scheduling algorithm $KS(W_1, 2)$, and TK_2^i are the output words of the key scheduling algorithm $KS(W_2, 1)$. For Deoxys-BC-256-128, the size of KT is 384 bits, there are three words W_1, W_2, W_3, and TK_1^i are outputs of $KS(W_1, 4)$, TK_2^i are the outputs of $KS(W_2, 2)$, and TK_3^i are outputs of $KS(W_3, 1)$. We note that the second variable to the KS functions are different on purpose, such that the more frequently updated parameter in the implementation does not suffer for a field multiplication. This parameter is the tweak input and has an α coefficient equals to 1 at most as possible.

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-th key schedule constants of the `AES`. We report their actual values in Appendix A.2.

Chapter 3

Security Claims

We provide our security claims for the different variants of Deoxys in Table 3.1. We recall that the variants are defined in part by the bit size k of key and the bit size t of the tweak in Section 2.2. We give the security goals expressed in terms of these two parameters as they are the same for all the variants within the same mode.

One can see that we do claim full k-bit security for $\texttt{Deoxys}^{\neq}$ for a nonce-respecting user, in contrary to other modes like AES-GCM [22] or OCB3 [20], which only ensure birthday-bound security. Even though we only claim birthday-bound security concerning $\texttt{Deoxys}^{=}$ in the nonce-respecting user model, we conjecture that full security can be reached.

	Security (bits)	
Goal (nonce-respecting user)	$\texttt{Deoxys}^{\neq}$	$\texttt{Deoxys}^{=}$
Confidentiality for the plaintext	k	$n/2$
Integrity for the plaintext	n	$n/2$
Integrity for the associated data	n	$n/2$
Integrity for the public message number	n	$n/2$

	Security (bits)	
Goal (nonce-misuse user)	$\texttt{Deoxys}^{\neq}$	$\texttt{Deoxys}^{=}$
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	$n/2$

Table 3.1: Security goals of Deoxys. The upper table stands for the situation where the user will never repeat the same value N for the same key (nonce-respecting user). The lower table stands for the situation where such repetitions in N for the same key are allowed (non-misuse user). There is no secret message number.

In the table we assume the public message number is a nonce and there is no secret message number. We also assume that the total size of the associated data and the message does not exceed $16 \cdot 2^{max_l}$ bytes, thus 2^{65} bytes for all variants of Deoxys. Moreover, the maximum number of messages that can be handled for a same key is 2^{max_m}, that is 2^{64} for all variants of Deoxys.

Chapter 4

Security Analysis

The Advanced Encryption Standard (AES) and AES-type ciphers in the past two decades 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. A progress in analysis has been introduced in the past several years, and as a rule, 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 [3,4] and the Meet-in-the-Middle (MITM) attacks [9,11,12,21] 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.

4.1 Differential Attacks

Designing a cipher resistant against single-key differential attacks is fairly simple and can be done by carefully choosing the diffusion layer (ensure that the branch number is high enough). For resistance against related-key differential attacks, we still do not have such a simple strategy. We do have, however, search algorithms and tools [5,6,13,14,24,27] that given a key schedule can return the upper bound on 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. We use precisely these algorithms in our security analysis against related-key attacks.

These 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 [5], split approach [6], and extended split approach [13]. We omit the details on how these search algorithms operate due to their complexity, and further, give only the final results produced by the tools.

In Table 4.1, we give the results for our tweakable block cipher Deoxys-BC where the sum of the key and tweak sizes is 256 bits. Assuming that each of the active S-Boxes can reach the maximal differential probability of the AES S-Box $p_{max} = 2^{-6}$, we get the upper bound given in the third column. Using at least $r = 10$ rounds for this cipher, the number of active S-Boxes is lower-

rounds	**active S-Boxes**	**upper bound on probability**	**method used**
1	0	2^{0}	trivial
2	0	2^{0}	trivial
3	1	2^{-6}	Matsui's
4	5	2^{-30}	Matsui's
5	9	2^{-54}	Matsui's
6	12	2^{-72}	Matsui's
8	≥ 17	2^{-108}	extended split (4R+4R)
10	≥ 22	2^{-132}	extended split (5R+5R)

Table 4.1: `Deoxys-128-128`: Upper bounds on probability of the best round-reduced related-key related-tweak differential characteristics when $k = 128$ and $t = 128$.

rounds	**active S-Boxes**	**upper bound on probability**	**method used**
1	0	2^{0}	trivial
2	0	2^{0}	trivial
3	0	2^{0}	trivial
4	1	2^{-6}	Matsui's
5	4	2^{-24}	Matsui's
6	8	2^{-48}	Matsui's
12	≥ 22	2^{-132}	extended split (6R+6R)

Table 4.2: `Deoxys-256-128`: Upper bounds on probability of the best round-reduced related-key related-tweak differential trails.

bounded by 22, meaning that the probability of the associated differential trail is upper-bounded by $2^{-6\times 22} = 2^{-132}$, thus such trails cannot be exploited due to the state size of 128 bits (and the fact that the attacker cannot construct more than 2^{128} plaintext pairs to start the attack).

We perform the same analysis when the sum of the key and tweak sizes is 384 bits and the results from Table 4.2 show that we reach more than $\lceil \frac{128}{6} \rceil = 22$ S-Boxes after 12 rounds.

Thus each of these ciphers has a security margin of at least four rounds (recall that the first cipher has 14 rounds, while the second 16). This makes them highly resistant against related-key related-tweak attacks.

4.2 Meet-in-the-Middle Attacks

Additionally, we scrutinize the resistance of our design in regard to the recent advanced meet-in-the-middle attack on `AES` conducted in [11]. 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 the design we propose introduces a new `AES` schedule, we have analyzed how it interacts with the `AES` round function.

For a given tweak value, `Deoxys-BC` behaves as the `AES` with a new schedule with partially known values (subtweaks) XORed between each round, without additional input values. This key schedule is fully linear as it first applies implements a byte permutation and the multiplies each of the 16 bytes of the state by 2 in $GF(256)$. In that context, a first analysis shows that the meet-in-the-middle technique from [11] can attack up to 8 rounds, where the AES key schedule for 128-bit keys stops the attack at 7 rounds.

4.3 Security Against Other Attacks

As mentioned earlier in the chapter, the security bound of Deoxys-BC 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-BC as on AES. This gives us a security reduction from AES, however, we note that as Deoxys-BC has more rounds, the security margin is higher.

The type of attacks we also have to investigates (besides the above two) are the ones that rely on some additional property of the AES key schedule that might be missing in Deoxys-BC. In particular the constants in the key schedule of AES in fact stop several other attacks: slide [7], rotational [19] and the internal differential attacks [25]. However, the schedule of Deoxys-BC also applies constants, thus there is no threat coming from these types of attacks.

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 [10, 15, 17] using this extra tweak input). However, we emphasize that we do not claim any resistance of Deoxys-BC in this attack model.

Chapter 5

Features

The starting point of our design is to provide a sound ad-hoc tweakable block cipher based on AES. Indeed, the main idea heavily exploited in the design of Deoxys is the introduction of an efficient tweakable block cipher Deoxys-BC, belonging to the family of the well-known AES-based primitives. Speed benchmarks show that Deoxys achieves almost the same speed as OCB while offering more security than OCB.

Deoxys-BC is a secure instantiation of a more general framework (TWEAKEY [18]) and does not rely on big field multiplications as previous tweakable ciphers proposals. Structurally, Deoxys-BC can be seen as a standalone primitive, whereas previous attempts at building tweakable block ciphers use a given block cipher as a black box and use it in a particular mode. The design is in particular very efficient on latest Intel processors, where we can reach 1.32 cpb for scenario with nonce-respecting users. We detail more precisely below the main features on Deoxys.

- Deoxys has a good security margin for all the recommended parameters. We measure the security margin in terms of number of rounds: the small variant of Deoxys counts 14 rounds and the large variant 16 rounds. The best known attacks on AES-based design in secret-key models for similar size of keys reach 7 to 9 rounds. For a stronger adversarial model in the related-key scenario, the two large variants of the AES are theoretically broken by known results from [3, 4]. With 14 rounds or more, the two versions of Deoxys offer a confident security margin regarding this class of attacks. Interestingly, Deoxys-BC-256-128 is very similar to AES-256 for a fixed tweak value, but we have shown that a good key schedule can significantly reduce the number of rounds required for a secure cipher: Deoxys-BC-256-128 only needs 12 rounds to be secure to a related-key attack, whereas AES-256 on 14 rounds is already subject to a theoretical related-key distinguisher.

- The security arguments of Deoxys are directly inherited from the two modes used in our design. Indeed, for nonce-respecting users, Deoxys benefits from the proofs of OCB3 mode, while for nonce-repeating users, we designed a mode generalizing COPA [1] to tweakable block ciphers.

- Deoxys achieves good software performances for software implementations. As most of the AES-based designs, it hugely benefits from the new AES-NI instruction set added in the latest processors. In addition, as we use fully parallelizable modes, the cycles per byte count drops significantly – the current speed of our nonce-respecting design is faster than AES-GCM on the Intel Sandy Bridge (although AES-GCM ensures only birthday-bound security).

- Deoxys is efficient for small messages. This is due again to the fact that we use a tweakable block cipher: it allows to avoid any precomputation (like in OCB or AES-GCM). The first 128-bit message block is handled directly, and taking in account the tag generation one needs $m + 1$ internal cipher calls to process messages of m block of n bits each. This is particularly important in many lightweight applications where message sent are usually composed of a few dozens of bytes (this is common disadvantage of sponge-based or stream cipher based lightweight designs like FIDES [2]) or ALE [8]).

- Deoxys also benefits from the vast research literature on the cryptanalysis of the AES. In effect, being an AES-based primitive, the tweakable block cipher Deoxys-BC is subject to the same class of attacks than AES, which consists of an active research line since 15 years.

- Deoxys is simple for both the construction of the internal tweakable block cipher and for the authentication mode. It uses well-studied building blocks and is arguably easy to analyze. The implementation of Deoxys is also easy, and can reuse the design strategies, implementations and optimizations available for the AES.

- Deoxys can resists to side-channel attacks with the same techniques as AES. Literature in this area available for the AES can be very easily adapted to the case of any AES-based designs, including Deoxys.

- Deoxys has smooth parameters handling. We define some recommended parameter sets in this document, but any user can pick its own variant of the inner tweakable block cipher Deoxys-BC by adapting the key and tweak sizes at his/her convenience. This flexibility comes from the unified vision of the key and tweak material brought by the TWEAKEY framework. It means that one implementation of the cipher is sufficient to support all versions with different key and tweak sizes (with the same cumulative size). This feature extends to the whole Deoxys design.

Chapter 6

Design Rationale

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 loose this feature when instantiated with the current constructions that use 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 would enable us to reach full 128-bit security for both confidentiality and authenticity. This direction of research was not explored yet as it was believed that ad-hoc tweaking of `AES`-like ciphers is not an easy task from points of view of both security and efficiency (adding some extra freedom to the attacker seems to enable more powerful attacks and thus implies many more rounds). As our design is lightweight, we are also careful in choosing the internal permutations of the cipher and the mode that provides the authenticated encryption.

6.1 Details for the `STK` construction

Designing a secure round function for block ciphers has become a fairly easy task – a S-Box layer and a diffusion layer based on MDS code immediately provide good security margins 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 as possible linear operations).

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 schedule, and the use of finite field multiplications, in a related-key attack the differences in the nibbles of the subkeys cannot cancel each other too many times (in subkeys of different rounds), and thus security against these type of attacks can be proven when then number of rounds is not necessarily very high.

We denote with `TK`-p the cipher where the master key is p times larger than the state (and thus is divided into p keys). In our proposals, we work only with `TK`-2, and `TK`-3, but the same strategy applies when $p > 3$. Let us choose an arbitrary position of a nibble at the beginning of each of the p key schedules. For instance, we fix the position $(1, 1)$ and we investigate how the p nibbles 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 permutations, the initial p nibbles 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 nibble 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 constants α_i, $i = 1, \ldots, p$) produces r tuples $[x_1^1, \ldots, x_p^1], \ldots, [x_1^r, \ldots, x_p^r]$, such that $x_{j+1}^k = \alpha_i \cdot x_j^k$. All of them are integrated to the internal state by considering the $r+1$ XOR values $\bigoplus_{i=1}^p x_i^k$, for $0 \leq k \leq r$. Those values incorporated to the internal state can be rewritten by using the following $p \times (r+1)$ Vandermonde matrix

$$\mathbf{V} = \left(\alpha_i^j\right)_{i,j} = \begin{pmatrix} \alpha_1^0 & \alpha_1^1 & \ldots & \alpha_1^r \\ \alpha_2^0 & \alpha_2^1 & \ldots & \alpha_2^r \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_p^0 & \alpha_p^1 & \ldots & \alpha_p^r \end{pmatrix},$$

by a right-matrix multiplication: $\mathbf{y} = \mathbf{x} \times \mathbf{V}$, for $\mathbf{x} = [x_1^0, \ldots, x_p^0]$. To minimize the number of non-zero elements in $\mathbf{y}$ for $\mathbf{x} \neq 0$, we need to ensure that all the rows in $\mathbf{V}$ are linearly independent. This is true as long as the α_i coefficients, $1 \leq i \leq p$ are pairwise distinct.

As a result, cancellation of values (and differences in general as the key schedule is linear) in the chosen nibble of TK-p cannot occur more than $p-1$ times. For TK-2 this means that the cumulative difference coming from the subkeys can be canceled only once by XOR of the subkeys (but the remaining will have a difference). For TK-3, this cancellation event can happen twice.

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 steps that follow are the choice of the permutation and the choice of the constants α's. The choice of permutation was done after trying several of them – we settled down on the one that provides security against related-key attacks in the least number of rounds (we inspected the security with the tools specified in Section 4). The constants were chosen to make the construction lightweight implementation-friendly.

6.2 From Block Cipher to Tweakable Block Cipher

The STK construction (with specified permutation and α's) 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 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-BC-128-128). A similar transition was made from the TK-3 block cipher (with 384-bit tweakey) to Deoxys-BC-256-128.

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: 1) the key schedule is linear, it never has any active S-boxes, and 2) 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).

Chapter 7

Software Benchmark

As `Deoxys` is based on the `AES`, it allows very efficient software implementation on the processors that support `AES-NI`. In addition, the mode allows complete parallelizations of the `AES-NI` calls. The actual overhead compared to `AES` comes from the increased number of rounds and 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, the key schedule uses multiplication by a low hamming weight constant, 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`.

We used the two Intel processor families Intel Sandy Bridge and Intel Haswell with `AES-NI` enabled to obtain benchmarks for `Deoxys`. The code was compiled on Linux with `gcc v4.8.1`. The reported speed was taken as an average over multiple execution of the code with the same fixed message length. We did take into account the key schedule as well as the loading the bytes from the memory and storing them back to memory.

Design	**128B**	**256B**	**512B**	**1024B**	**2048B**	**4096B**
$\texttt{Deoxys}^{\neq}$`-128-128`	2.30	1.73	1.45	1.36	1.15	1.13
$\texttt{Deoxys}^{\neq}$`-256-128`	4.26	2.53	1.92	1.57	1.48	1.32
$\texttt{Deoxys}^{=}$`-128-128`	4.50	3.42	2.84	2.61	2.43	2.33
$\texttt{Deoxys}^{=}$`-256-128`	7.89	5.13	3.55	3.07	2.75	2.59

Table 7.1: Benchmarks for $\texttt{Deoxys}^{\neq}$ expressed in cycles per byte on `AES-NI` enabled Intel Sandy Bridge.

Design	**128B**	**256B**	**512B**	**1024B**	**2048B**	**4096B**
$\texttt{Deoxys}^{\neq}$`-128-128`	2.25	1.84	1.64	1.55	1.49	1.46
$\texttt{Deoxys}^{\neq}$`-256-128`	3.68	2.66	2.14	1.88	1.76	1.69
$\texttt{Deoxys}^{=}$`-128-128`	4.07	3.43	3.12	2.97	2.89	2.85
$\texttt{Deoxys}^{=}$`-256-128`	5.68	4.44	3.82	3.51	3.36	3.28

Table 7.2: Benchmarks for $\texttt{Deoxys}^{\neq}$ expressed in cycles per byte on `AES-NI` enabled Intel Haswell.

The results of the benchmarks of the encryption and authentication (simultaneously) of `Deoxys` are given in Tables 7.1,7.2. We can see that our nonce-respecting mode $\texttt{Deoxys}^{\neq}$ performs well, and is faster than AES-GCM on Intel Sandy Bridge. The second mode is around twice slower as it requires twice more calls to the cipher. As expected, the cycle per byte count in the first mode stabilizes after the message length reaches 256-512 bytes or more – this is due to the fact that no preprocessing step is required to start the cipher calls.

Although the above benchmark rely on the AES-NI instruction set, the simplicity of the 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 in this case compared to `AES`, will be very small, and the speed of `Deoxys-BC` will be very close to the speed of `AES`.

Chapter 8

Hardware Performances

Due to time constraints, we do not provide hardware implementations of `Deoxys`. However, we briefly explain in this chapter why we believe `Deoxys` would be a potentially rather lightweight candidate and the logic behind our ASIC implementation estimation.

Our starting point is the best ASIC lightweight implementation [23] of `AES`, that only requires 2400GE (from which 70% comes from the memory to store the key and the internal state). Since `Deoxys-BC` is exactly the `AES`, except the key and tweak layers, it is safe to estimate that the overhead will be due to storing the 128-bit tweak value and XORing it to the internal state (and in addition an extra 128-bit key for `Deoxys-BC-256-128`). In particular, the key schedule in `Deoxys-BC` seems to be more lightweight than the `AES` one, since it only requires wiring and a few simple multiplications in $GF(2^8)$. Therefore, we expect an overhead of about $128 \times (4.67 + 2.67) = 940$ GE for `Deoxys-BC`-128-128, by counting 2.67 GE per XOR (which might sometimes be optimized to 2.33 GE) and 4.67 GE for single-bit input flip-flop to store the tweak. We would need approximatively double this amount in the case of `Deoxys-BC-256-128`. Therefore, we estimate that the entire `Deoxys-BC-128-128` can be implemented with around 3400 GE, and `Deoxys-BC-256-128` with 4400 GE.

Concerning the authenticated encryption mode for `Deoxys`$^{\neq}$, one can remark that it calls directly `Deoxys-BC` on the incoming message blocks and directly outputs the corresponding ciphertext blocks. However, one needs to take in account the following three main potential overhead costs:

- a 128-bit checksum needs to be computed and stored. Therefore, one should count an additional $128 \times (4.67 + 2.67) = 940$ GE.
- the tweak value needs to be increased every `Deoxys-BC` call, and this operation can be quite expensive, because the carries needs to be taken care of. Comparing with other implementations, we estimate to 4 GE per bit for an integer addition when minimal area is the implementation goal. In our case, the addition is done on $max_l = 61$ bits, hence $61 \times 4 = 244$ GE. We note that using a different tweak update function (for example using an LFSR) would drastically reduce this cost without changing the security aspects of `Deoxys`$^{\neq}$ (we only need that the counter runs through all the possible values, the ordering does not matter).
- in case where associated data is input, one can see that a 128-bit authentication value needs to be maintained until the end, similarly to the checksum, and this would add another $128 \times (4.67 + 2.67) = 940$ GE. However, this can be simply avoided if the associated data is computed after the message blocks (by directly XORing the output of the `Deoxys-BC` calls to the checksum register). Therefore, we do not add extra cost for this part.

All in all, we estimate that `Deoxys-128-128` should be able to fit in about 4600 GE, and `Deoxys-256-128` in about 5600 GE. We emphasize that these are only very rough and possibly optimistic estimations and only real implementations will be able to confirm them.

We note that one very good advantage for `Deoxys` in hardware applications is that the speed overhead for small messages is null. Indeed, the very first message block is ciphered directly, without any precomputation. In RFID applications where only small data is likely to be protected (like a 96-bit Electronic Product Code), this will have a huge impact compared to sponge based or stream cipher based lightweight proposals that usually requires a long initialization period.

Concerning `Deoxys`$^=$, the reasoning is exactly the same, except that the 128-bit temporary authentication value must be maintained. Therefore, an extra 940 GE will have to be taken in account.

Chapter 9

Intellectual Property

`Deoxys` is not patented and is free for use in any application. If any of this information changes, the submitter/submitters will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list. We note that since `Deoxys` uses a mode belonging to the generic Θ`CB3` framework, it is unclear if patents relative to `OCB` (such as United States Patent No. 7,046,802; United States Patent No. 7,200,227; United States Patent No. 7,949,129; United States Patent No.8,321,675) apply to our proposal.

Chapter 10

Consent

The submitter/submitters hereby consent to all decisions of the CAESAR selection committee regarding the selection or non-selection of this submission as a second-round candidate, a third-round candidate, a finalist, a member of the final portfolio, or any other designation provided by the committee. The submitter/submitters understand that the committee will not comment on the algorithms, except that for each selected algorithm the committee will simply cite the previously published analyses that led to the selection of the algorithm. The submitter/submitters understand that the selection of some algorithms is not a negative comment regarding other algorithms, and that an excellent algorithm might fail to be selected simply because not enough analysis was available at the time of the committee decision. The submitter/submitters acknowledge that the committee decisions reflect the collective expert judgments of the committee members and are not subject to appeal. The submitter/submitters understand that if they disagree with published analyses then they are expected to promptly and publicly respond to those analyses, not to wait for subsequent committee decisions. The submitter/submitters understand that this statement is required as a condition of consideration of this submission by the CAESAR selection committee.

Bibliography

[1] Andreeva, E., Bogdanov, A., Luykx, A., Mennink, B., Tischhauser, E., Yasuda, K.: Parallelizable and Authenticated Online Ciphers. In Sako, K., Sarkar, P., eds.: ASIACRYPT (1). Volume 8269 of Lecture Notes in Computer Science., Springer (2013) 424–443

[2] Bilgin, B., Bogdanov, A., Knezevic, M., Mendel, F., Wang, Q.: `FIDES`: Lightweight Authenticated Cipher with Side-Channel Resistance for Constrained Hardware. In Bertoni, G., Coron, J.S., eds.: CHES 2013. Volume 8086 of LNCS., Springer (August 2013) 142–158

[3] Biryukov, A., Khovratovich, D.: Related-Key Cryptanalysis of the Full AES-192 and AES-256. In Matsui, M., ed.: ASIACRYPT 2009. Volume 5912 of LNCS., Springer (December 2009) 1–18

[4] Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and Related-Key Attack on the Full AES-256. In Halevi, S., ed.: CRYPTO 2009. Volume 5677 of LNCS., Springer (August 2009) 231–249

[5] Biryukov, A., Nikolić, I.: Automatic Search for Related-Key Differential Characteristics in Byte-Oriented Block Ciphers: Application to AES, Camellia, Khazad and Others. In Gilbert, H., ed.: EUROCRYPT. Volume 6110 of Lecture Notes in Computer Science., Springer (2010) 322–344

[6] Biryukov, A., Nikolić, I.: Search for Related-Key Differential Characteristics in DES-Like Ciphers. In Joux, A., ed.: FSE. Volume 6733 of Lecture Notes in Computer Science., Springer (2011) 18–34

[7] Biryukov, A., Wagner, D.: Slide Attacks. In Knudsen, L.R., ed.: FSE'99. Volume 1636 of LNCS., Springer (March 1999) 245–259

[8] Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: `ALE`: `AES`-Based Lightweight Authenticated Encryption. In: FSE. Lecture Notes in Computer Science (2013) *to appear.*

[9] Demirci, H., Selçuk, A.A.: A Meet-in-the-Middle Attack on 8-Round AES. In Nyberg, K., ed.: FSE 2008. Volume 5086 of LNCS., Springer (February 2008) 116–126

[10] Derbez, P., Fouque, P.A., Jean, J.: Faster Chosen-Key Distinguishers on Reduced-Round AES. In Galbraith, S.D., Nandi, M., eds.: INDOCRYPT 2012. Volume 7668 of LNCS., Springer (December 2012) 225–243

[11] Derbez, P., Fouque, P.A., Jean, J.: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In Johansson, T., Nguyen, P.Q., eds.: EUROCRYPT 2013. Volume 7881 of LNCS., Springer (May 2013) 371–387

[12] Dunkelman, O., Keller, N., Shamir, A.: Improved Single-Key Attacks on 8-Round AES-192 and AES-256. In Abe, M., ed.: ASIACRYPT 2010. Volume 6477 of LNCS., Springer (December 2010) 158–176

[13] Emami, S., Ling, S., Nikolić, I., Pieprzyk, J., Wang, H.: The resistance of PRESENT-80 against related-key differential attacks. Cryptography and Communications (2013) 1–17

[14] Fouque, P.A., Jean, J., Peyrin, T.: Structural Evaluation of AES and Chosen-Key Distinguisher of 9-Round AES-128. In Canetti, R., Garay, J.A., eds.: CRYPTO 2013, Part I. Volume 8042 of LNCS., Springer (August 2013) 183–203

[15] Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. [16] 365–383

[16] Hong, S., Iwata, T., eds.: FSE 2010. In Hong, S., Iwata, T., eds.: FSE 2010. Volume 6147 of LNCS., Springer (February 2010)

[17] Jean, J., Naya-Plasencia, M., Peyrin, T.: Improved Rebound Attack on the Finalist Grøstl. In Canteaut, A., ed.: FSE 2012. Volume 7549 of LNCS., Springer (March 2012) 110–126

[18] Jérémy Jean and Ivica Nikolić and Thomas Peyrin: Tweaks and Keys for Block Ciphers: the `TWEAKEY` Framework (2014) *Article in preparation.*

[19] Khovratovich, D., Nikolic, I.: Rotational Cryptanalysis of ARX. [16] 333–346

[20] Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. In Joux, A., ed.: FSE 2011. Volume 6733 of LNCS., Springer (February 2011) 306–327

[21] Li, L., Jia, K., Wang, X.: Improved Meet-in-the-Middle Attacks on AES-192 and PRINCE. Cryptology ePrint Archive, Report 2013/573 (2013)

[22] McGrew, D., Viega, J.: The Galois/Counter mode of operation (GCM). Submission to NIST. http://csrc. nist. gov/CryptoToolkit/modes/proposedmodes/gcm/gcm-spec. pdf (2004)

[23] Moradi, A., Poschmann, A., Ling, S., Paar, C., Wang, H.: Pushing the Limits: A Very Compact and a Threshold Implementation of AES. In Paterson, K.G., ed.: EUROCRYPT. Volume 6632 of Lecture Notes in Computer Science., Springer (2011) 69–88

[24] Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In Wu, C., Yung, M., Lin, D., eds.: Inscrypt. Volume 7537 of Lecture Notes in Computer Science., Springer (2011) 57–76

[25] Peyrin, T.: Improved Differential Attacks for ECHO and Grøstl. In Rabin, T., ed.: CRYPTO 2010. Volume 6223 of LNCS., Springer (August 2010) 370–392

[26] Ristenpart, T., Rogaway, P.: How to Enrich the Message Space of a Cipher. In Biryukov, A., ed.: FSE 2007. Volume 4593 of LNCS., Springer (March 2007) 101–118

[27] Sun, S., Hu, L., Wang, P.: Automatic Security Evaluation for Bit-oriented Block Ciphers in Related-key Model: Application to PRESENT-80, LBlock and Others. Cryptology ePrint Archive, Report 2013/676 (2013)

Appendix 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.

	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 A.1: 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	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

Table A.2: 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 $\mathcal{S}(\texttt{0x3F}) = \texttt{0x25}$.

A.2 AES RCON constants

The Table A.3 below gives the values of constants RCON used in the key scheduling algorithm of the AES.

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

Table A.3: The AES RCON constants used in the key scheduling algorithm. The constants are written on lines from left to right, from top to bottom. For example, RCON[1] = 0x2f, RCON[2] = 0x5e, and RCON[11])=0xb3.