KIASU 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 17, 2014

Chapter 1

Introduction

In this note, we propose KIASU, a new authenticated encryption design based on a tweakable block cipher KIASU-BC that uses the well-studied AES round function as a building block. We plug this cipher into two different authenticated encryption modes inspired from fully parallel and provably secure modes OCB3 [24] and COPA [2] respectively.

In short, KIASU is a very fast authenticated encryption scheme in software (much faster than AES-GCM [28] and about the same speed as OCB3 [24]) that provides full security beyond the birthday bound (in contrary to AES-GCM [28] or OCB3 [24]) for both privacy and authenticity. Moreover, KIASU performs extremely well even 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 KIASU is made very simple as a tweakable block cipher is a very handy primitive to build an authenticated encryption scheme. Finally, KIASU can be lightweight (using existing AES lightweight implementation, the extra area mainly consisting in some extra bits of memory for the mode and to store the tweak), it has a very simple description and KIASU-BC is backward compatible with AES (by just using a tweak equal to zero). Actually, KIASU-BC is the first attempt to build an ad-hoc tweakable AES.

Organization of the document. In Chapter 2, we provide the specification of our proposal KIASU, which includes the descriptions of the internal tweakable block cipher KIASU-BC and the authentication mode we use. In Chapter 3, we precise the security claims for different classical scenarios, and we perform a security analysis regarding this proposal in Chapter 4. In Chapters 5 and 6, we analyze both software and hardware performances. In Chapters 7 and 8, we detail the 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 KIASU. We first give the recommended parameter sets and then proceed with the description of the design. We explain the two authenticated encryption modes KIASU$\neq$ and KIASU=, and then we describe KIASU-BC, a tweaked version of the AES block cipher. KIASU-BC is basically built by simply reusing the plain AES round function and by inserting a 64-bit tweak in each of the $r = 10$ rounds. This can be seen as an extended version of the key schedule from the AES where the subkeys are XORed with the tweak value before each AddRoundKey operation.

We first introduce some notations. We denote $E_K(T, P)$ the enciphering of the n-bit plaintext P with the tweakable block cipher KIASU-BC with k-bit key K and t-bit tweak T (similarly, D represents the deciphering process). Note that we have $n = 128$, $k = 128$ and $t = 64$ for KIASU-BC. 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 KIASU 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 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 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 $max_l = 2^{\lceil t/2 \rceil - 3} = 2^{29}$ and $max_m = 2^{\lfloor t/2 \rfloor} = 2^{32}$. 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. Note that there is a tradeoff possible here between max_l and max_m, as long as $max_l \cdot \max_m = 2^{t-3}$.

2.1 Parameters

The only parameter to the authenticated encryption KIASU is the choice between a nonce-respecting mode (denoted with a $\neq$ sign) or a nonce-misuse resistant mode (denoted with a = sign). No parameter affects the internal tweakable block cipher KIASU-BC, which always uses a 128-bit key and a 64-bit tweak to select a permutation on 128 bits, mapping the plaintext space to the the ciphertext space. As we mention explicitly in the next chapter, selecting a different mode results in

quite different security notions.

2.2 Recommended Parameter Sets

We propose two designs that have the same key, tag and public message number lengths, but differ in the security goals:

- KIASU$\neq$: 128-bit key K, 128-bit tag `tag`, 32-bit public message number N. This design is made for nonce-respecting users (i.e. it is assumed that the public message number is always different for a same key). The encryption/authentication algorithm is denoted as $\mathcal{E}^{\neq}$, while the decryption/verification as $\mathcal{D}^{\neq}$.
- KIASU=: 128-bit key K, 128-bit tag `tag`, 32-bit public message number N. This design is made for nonce-misuse users (i.e. it is preferable, but not required that the public message number is always different for a same key). The encryption/authentication algorithm is denoted as $\mathcal{E}^{=}$, while the decryption/verification as $\mathcal{D}^{=}$.

The first parameter set is our preferred one, where the public message number N is a nonce.

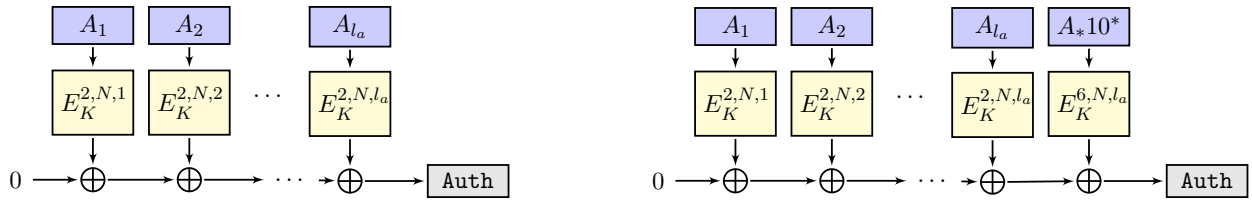
2.3 Authenticated Encryption

In this section, we provide the high-level description of our proposal. KIASU uses a tweakable block cipher KIASU-BC as internal primitive (specified in Section 2.4), and we describe here the simple authenticated encryption modes built on top of it. KIASU has two main 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 ΘCB3 [24], 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 [2], 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 ΘCB3 [24] 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.

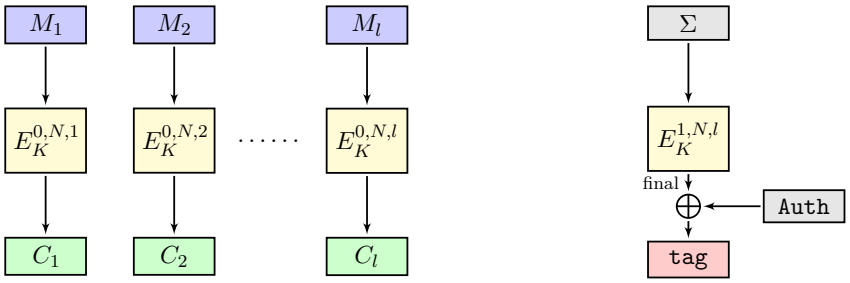


Figure 2.2: Message processing: in the case where the message-length is a multiple of the block size: no padding needed.

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 */*
$A_1||\ldots||A_{l_a}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$
Auth $\leftarrow 0^n$
for $i = 1$ **to** l_a **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 */*
$M_1||\ldots||M_l||M_* \leftarrow M$ 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 $\leftarrow E_K(001||N||l,$ Checksum$)$
  $C_* \leftarrow \epsilon$
else
  Checksum $\leftarrow$ Checksum $\oplus\, pad10^*(M_*)$
  Pad $\leftarrow E_K(100||N||l, 0^n)$
  $C_* \leftarrow M_* \oplus trunc_{|M_*|}(Pad)$
  Final $\leftarrow E_K(101||N||l,$ Checksum$)$
end

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

return $(C_1||\ldots||C_l||C_*,$ tag$)$

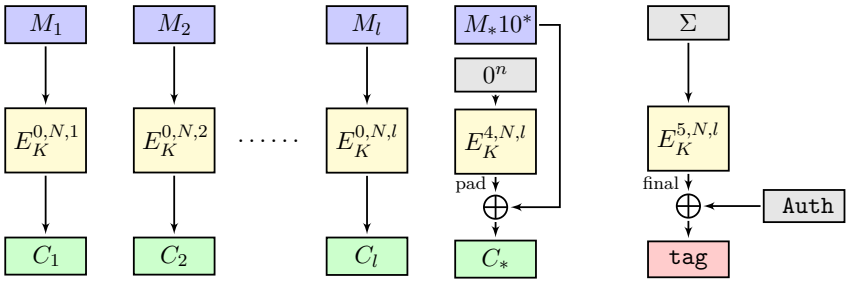


Figure 2.3: Message processing: in the case where the message-length is a not multiple of the block size. Note that the checksum Σ is computed with a 10^* padding for block M^*.

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 */*
$A_1||\ldots||A_{l_a}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$
Auth $\leftarrow 0^n$
for $i = 1$ **to** l_a **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 */*
$C_1||\ldots||C_l||C_* \leftarrow C$ 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 $\leftarrow E_K(001||N||l,$ Checksum$)$
  $M_* \leftarrow \epsilon$
else
  Pad $\leftarrow E_K(100||N||l, 0^n)$
  $M_* \leftarrow C_* \oplus trunc_{|C_*|}$(Pad)
  Checksum $\leftarrow$ Checksum $\oplus\, pad10^*(M_*)$
  Final $\leftarrow E_K(101||N||l,$ Checksum$)$
end

/ Tag verification */*
$\texttt{tag}' \leftarrow trunc_\tau$(Final $\oplus$ Auth)
if $\texttt{tag}' = \texttt{tag}$ **then**
  return $(M_1||\ldots||M_l||M_*)$
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 [2] 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 [2], where tag splitting is used for messages with $|M| < n$ and XLS [32] for messages with $|M| > n$.

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

Figure 2.4: Handling of the associated data for the nonce-respecting mode with a PMAC-like processing.

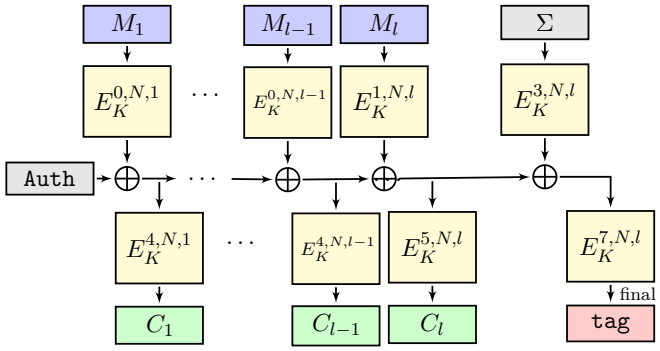


Figure 2.5: Message processing: in the case where the message-length is a multiple of the block size: no padding needed.

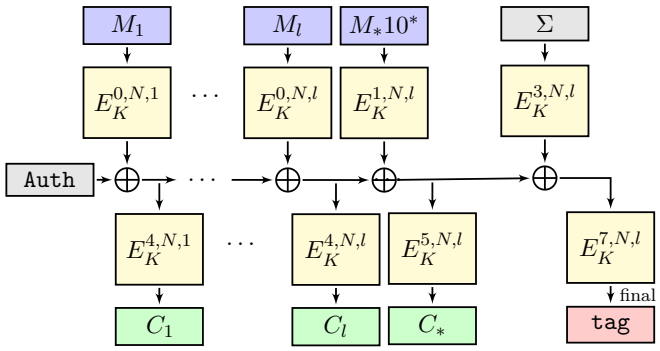


Figure 2.6: Message processing: in the case where the message-length is a not multiple of the block size. Note that the checksum Σ is computed with a 10^* padding for block M^*.

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 $|A_*| < n$
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 $\leftarrow E_K(110||N||l_a, \text{Auth})$

/* *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** $l - 1$ **do**
  Checksum $\leftarrow$ Checksum $\oplus\, M_i$
  Auth $\leftarrow$ 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**
  Checksum $\leftarrow$ Checksum $\oplus\, M_l$
  Auth $\leftarrow$ Auth $\oplus\, E_K(000||N||l, M_l)$
  $C_l \leftarrow E_K(100||N||l, \text{Auth})$
  $M_* \leftarrow pad10^*(M_*)$
  Checksum $\leftarrow$ Checksum $\oplus\, M_*$
  Auth $\leftarrow$ Auth $\oplus\, E_K(001||N||l, M_*)$
  $C_* \leftarrow E_K(101||N||l, \text{Auth})$
else
  Checksum $\leftarrow$ Checksum $\oplus\, M_l$
  Auth $\leftarrow$ 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* */
Auth $\leftarrow$ Auth $\oplus\, E_K(011||N||l, \text{Checksum})$
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||...||A_{l_a}||A_* ← A where each |A_i| = n and |A_*| < n
Auth ← 0^n
for i = 1 to l_a − 1 do
    Auth ← Auth ⊕ E_K(010||N||i, A_i)
end
if A_* ≠ ϵ then
    Auth ← Auth ⊕ E_K(010||N||l_a, A_{l_a})
    Auth ← Auth ⊕ pad10*(A_*)
else
    Auth ← Auth ⊕ A_{l_a}
end
Auth ← E_K(110||N||l_a, Auth)

/* Ciphertext */
C_1||...||C_l ← C where each |C_i| = n
Checksum ← 0^n
for i = 1 to l − 1 do
    X_i ← D_K(100||N||i, C_i)
    M_i ← D_K(000||N||i, Auth ⊕ X_i)
    Checksum ← Checksum ⊕ M_i
    Auth ← X_i
end
X_l ← D_K(101||N||l, C_l)
M_l ← D_K(001||N||l, Auth ⊕ X_l)
Checksum ← Checksum ⊕ M_l
Auth ← X_l
M_l ← unpad10*(M_l)

/* Tag verification */
Auth ← Auth ⊕ E_K(011||N||l, Checksum)
Final ← E_K(111||N||l, Auth)
tag' ← trunc_τ(Final)
if tag' = tag then
    return (M_1||...||M_l)
else
    return ⊥
end
```

2.4 KIASU-BC: a Tweakable Block Cipher

For encryption, the block cipher KIASU-BC denoted E takes three inputs: a 64-bit tweak T, a 128-bit key K and a 128-bit plaintext P. It outputs a 128-bit ciphertext $C = E_K(T, P)$ as the encryption of P under the key K for the tweak value T. Similarly, for decryption, the ciphertext C is mapped back to the plaintext by $E_K^{-1}(T, C) = P$.

In short, KIASU-BC is *exactly* the AES cipher, but with a fixed 64-bit tweak value XORed to the internal state (on the two first rows) after each round key addition. There is no tweak schedule. Our cipher can actually be seen as one of the simplest instance of the more general TWEAKEY framework [21] (see Figure 2.7). As it is based on AES, the 128-bit internal state of KIASU-BC can be viewed as a 4×4 matrix of bytes in the field noted $\mathbb{K}$ defined as $GF(256)$ by the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. A note about 16 bytes to 4×4 matrix conversation. To load the bytes to the state of the cipher, in accordance to the load/store instructions on the Intel processors, the bytes are loaded from the first column to the last, and from the top row to the bottom row. That is, the first byte is stored at position $(1, 1)$ of the matrix, the second at $(2, 1)$, the fifth at $(1, 2)$, ..., the last at $(4, 4)$. Similarly are loaded the 8 bytes of the tweak but only at the two two rows, i.e. the first at $(1, 1)$, then $(2, 1), (1, 2), (2, 2), \ldots, (2, 4)$.

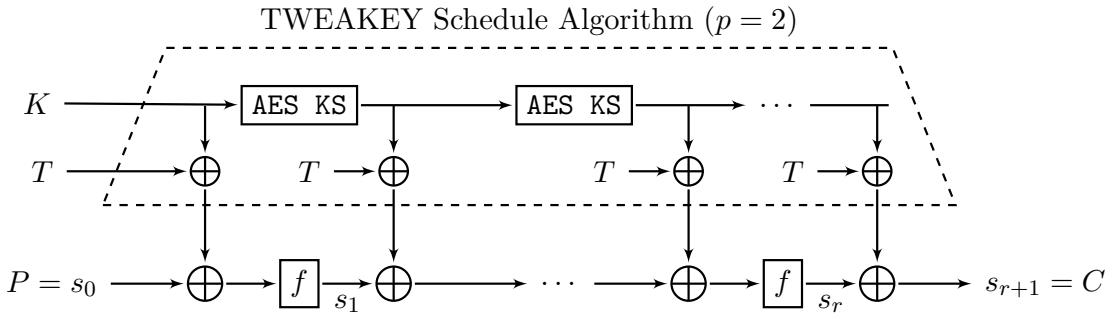


Figure 2.7: Instantiation of the TWEAKEY framework for KIASU-BC.

2.4.1 Encryption

KIASU-BC reuses the entire AES round function f[1]. This round function is composed of four steps SubBytes, ShiftRows, MixColumns, and AddRoundTweakey in this order (slightly modifying the FIPS 197 terminology in [1]):

- SubBytes: applies a Sbox $\mathcal{S}$ on each of the 16 bytes of the internal state (see Appendix A.1).
- ShiftRows: rotates bytes located in row i by i positions to the left in the state byte matrix
- MixColumns: applies to each byte column a column-wise linear layer defined by the multiplication in $\mathbb{K}$ with the Maximum-Distance-Separable (MDS) matrix $\mathbf{M}$:

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

- AddRoundTweakey: XOR the 128-bit round key to the internal state and the 64-bit tweak T to the two first rows (see Figure 2.8).

The internal state is initialized to P and KIASU-BC (as AES) applies 10 such f rounds in total, with in addition an AddRoundTweakey layer before the very first round. Moreover, as in AES the last

[1] In terms of AES-NI instructions on the latest processors, we use aesenc and aesenclast as round functions (without any additional modifications!).

$$T = \begin{array}{|c|c|c|c|} \hline T_0 & T_2 & T_4 & T_6 \\ \hline T_1 & T_3 & T_5 & T_7 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Figure 2.8: Tweak in `KIASU-BC`: the 64-bit values of the tweak $T = T_0||T_1 \ldots ||T_7$ are placed on the top two rows of the `AES` internal state.

of the 10 rounds does not have the `MixColumns` layer and the output of this last round represents the ciphertext.

As in `AES`, the first round key is set to K and the next one K_{i+1} is generated with the following function from the previous one K_i, $0 \leq i$. We denote $K_i[j]$ the j-th byte of the subkey K_i, where the byte are numbered from left to right and up to bottom, and `RCON` the constants of the `AES` key schedule (see Appendix A.2 for the actual values). Graphically, this key derivation is also described on Figure 2.9.

$$\begin{cases} K_i[j] &= K_i[j-4] + K_{i-1}[j] & \text{for } j = 4, \ldots, 15 \\ K_i[0] &= K_{i-1}[0] + \mathcal{S}(K_{i-1}[13]) + \texttt{RCON}[i] \\ K_i[1] &= K_{i-1}[1] + \mathcal{S}(K_{i-1}[14]) \\ K_i[2] &= K_{i-1}[2] + \mathcal{S}(K_{i-1}[15]) \\ K_i[3] &= K_{i-1}[3] + \mathcal{S}(K_{i-1}[12]). \end{cases}$$

We emphasize that apart from the additional tweak XOR operations between each round, the `AES` encryption process stays exactly the same, including the key scheduling algorithm. In particular, when the tweak equals zero, we have that $\texttt{KIASU-BC}_K(0, P) = \texttt{AES}_K(P)$ for any K and P.

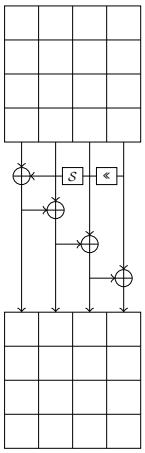


Figure 2.9: The original `AES-128` key scheduling algorithm. One cell represents one byte, the « performs a rotation upwards by one cell of the whole 4-byte column, and $\mathcal{S}$ is the `AES` non-linear S-Box.

2.4.2 Decryption

The decryption round function f^{-1} is composed of four steps `InvAddRoundTweakey`, `InvMixColumns`, `InvShiftRows`, `InvSubBytes` in this order. We note that the same tweak value T is used for both encryption and decryption. Since we are transforming back the ciphertext into the plaintext, the order of the operations described for the encryption side are performed in reverse order. Consequently, in the first round, the operation `InvMixColumns` is omitted. Note also that the subkeys are used in reverse order.

- `InvAddRoundTweakey`: XOR the 128-bit round key to the internal state and the 64-bit tweak T to the two first rows.
- `InvMixColumns`: applies to each byte column a column-wise linear layer defined by the multiplication in $\mathbb{K}$ with the Maximum Distance Separable (MDS) matrix (which is the inverse of the `MixColumns` diffusion matrix):

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

- `InvShiftRows`: rotates bytes located in row i by i positions to the right in the state byte matrix
- `InvSubBytes`: applies the inverse Sbox $\mathcal{S}^{-1}$ on each of the 16 bytes of the internal state (see Appendix A.1).

Chapter 3

Security Claims

We provide our security claims for KIASU in Table 3.1. One can see that we do claim full 128-bit security for KIASU≠, in contrary to other modes like AES-GCM [28] or OCB3 [24] which only ensure birthday-bound security. Even though we only claim birthday-bound security concerning KIASU= in the nonce-respecting user model, we conjecture that full 128-bit security can be reached.

We recall that we limit the number of messages that can be handled with the same key to $max_m = 2^{\lfloor t/2 \rfloor} = 2^{32}$, and each message can have at most $16 \cdot max_l = 2^{33}$ bytes. 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 unique. This also naturally implies that $|N| = \log_2(max_m) = 32$. Note that there is a tradeoff possible here between max_l and max_m, as long as $max_l \cdot \max_m = 2^{t-3}$.

	Security (bits)	
Goal (nonce-respecting user)	KIASU≠	KIASU=
Confidentiality for the plaintext	128	64
Integrity for the plaintext	128	64
Integrity for the associated data	128	64
Integrity for the public message number	128	64

	Security (bits)	
Goal (nonce-misuse user)	KIASU≠	KIASU=
Confidentiality for the plaintext	none	64
Integrity for the plaintext	none	64
Integrity for the associated data	none	64
Integrity for the public message number	none	64

Table 3.1: Security goals of KIASU. 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 N is a nonce and there is no secret message number. We also assume that the total length of the message (along with the length of the associated data) does not exceed 2^{33} bytes in both of the recommended parameter sets of KIASU.

Chapter 4

Security Analysis

In general, our design can be seen as an instantiation of secure authenticated encryption modes based on a new tweakable block cipher. However, the security goal of this cipher, unlike most of the tweakable block ciphers that only provide birthday bound security when plugged in the mode, is much higher – indeed we claim full 128-bit security against all possible attacks in the secret key model. Our claims are based on the strength of the AES and on a very careful study that no potentially exploitable weaknesses are introduced by the 64-bit tweak.

The AES and AES-like ciphers in the past two decades have been the object 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. Many advances in analysis has been introduced recently, and they involve in general a careful study of the key schedule of AES-like ciphers. In other words, the latest attacks extensively rely on how the key is processed in the rounds of the ciphers. Two such notable examples are the related-key differential attacks [4, 5] and the Meet-in-the-Middle (MITM) attacks [9, 12, 13, 25] on AES family of ciphers. These two family of attacks could pose a threat as well to other ciphers based on AES round function but with a different key schedule. For designers, adding a tweak to a block cipher is quite similar to adding extra key material and, as such, one should be particularly careful with how the attacker might exploit weaknesses in the tweak schedule as he can exploit weaknesses in the key schedule.

As KIASU-BC is based on AES, we focus our effort on evaluating the security of the cipher against the above two classes of attacks (related-key differential attacks the Meet-in-the-Middle attacks), with the very important specificity that we fully take the tweak input into consideration.

4.1 Differential Attacks

Designing an AES-like cipher resistant against single-key differential attacks is fairly simple and can be done by carefully choosing the diffusion layer: we ensure that the branch number is high enough so as to ensure sufficient number of active nibbles. For resistance against related-key differential attacks (in which the adversary is able to insert differences in the key input of the block cipher), adding subkeys every round makes this simple reasoning fail, and it is very hard to be able to prove a sufficient number of active Sboxes (attempts like LED block cipher [18] require an important number of rounds to achieve this). We do have, however, search algorithms and tools [6, 7, 15, 16, 30, 33] that given a key schedule can return the upper bound on the probability of the best related-key differential characteristics, and in the case where such a bound is low, practically provide and prove the resistance against simple related-key differential attacks. We use precisely these algorithms to assess the security of KIASU-BC against related-key attacks.

These tools have been designed to look for related-key characteristics. However, here we allow the adversary to operate in a stronger setting of related-key and/or related-tweak (in which the adversary is able to insert differences in the tweak input of the block cipher). 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 for 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 [6], split approach [7], and extended split approach [15]. We omit the details on how these search algorithms operate due to their complexity, and we will only provide the final results (refer to Table 4.1) produced by the tools. We note that the results were produced after several days of computing on a 64-core processor.

The best related-key related-tweak differential characteristics for 3 and 4 rounds of KIASU-BC have 1 and 8 active Sboxes, respectively. The search time complexity is exponential in all the input parameters, but it also depends on the number of rounds, hence finding the best characteristics for higher number of rounds becomes very time consuming. However, the tools can still provide lower bounds on the number of active Sboxes – we stress that these are lower bounds and the actual numbers might be much larger. The bounds produced by the tools for KIASU-BC are 14 active Sboxes for 5 rounds, and 22 active for 7 rounds. Hence, the probability of any characteristic on more than 6 rounds is not higher than $2^{-22 \cdot 6} = 2^{-132} < 2^{-128}$. Indirectly, this means that our tweakable block cipher, in the framework of related-key related-tweak differential attacks has only at most one round security loss compared to AES as any characteristic in AES on more than 5 rounds has a probability lower than 2^{-128}.

A similar observation holds if we take characteristics for boomerangs: in the case of KIASU-BC there are 7-round boomerangs (3 rounds + 4 rounds) with 18 active Sboxes, whereas for AES there are 6-round boomerangs (3 rounds + 3 rounds) with 20 active Sboxes.

Thus, the full 10-round KIASU-BC still has a comfortable security margin against differential attacks that exploit relations of keys and tweaks.

rounds	**active SBoxes**	**upper bound on probability**	**method used**
1	0	2^{0}	trivial
2	0	2^{0}	trivial
3	1	2^{-6}	Matsui's
4	8	2^{-48}	Matsui's
5	≥ 14	2^{-84}	Matsui's
7	≥ 22	2^{-132}	extended split (3R+4R)

Table 4.1: Upper bounds on probability of the best round-reduced related-key related-tweak differential characteristics.

4.2 Meet-in-the-Middle Attacks

We now mention the recent advances in the class of meet-in-the-middle attacks on the AES. In [10], Demirci and Selçuk introduced a distinguisher for 4 internal rounds of AES, allowing with some tradeoffs to construct a key-recovery attack on 7-round of AES-192 and AES-256. Then, Dunkelman, Keller and Shamir showed in [14] new techniques on top on the Demirci and Selçuk attack to decrease the memory requirements of the precomputation phase of the meet-in-the-middle, which allows to further extend the tradeoff and reach an attack on 7-round AES-128. Finally, Derbez, Fouque and Jean showed in [12] that the memory requirements are again smaller than estimated in [14], and they reach a smaller overall complexity of the meet-in-the-middle attacks on the three AES versions.

All this line of attacks strongly depends on the AES key schedule to guess key material and partially encrypts some well-crafted plaintexts/ciphertexts to check for a match in a precomputed table. To minimize the number of key bytes guesses, the adversary takes advantage of the linear relations existing in the AES key schedule to deduce more byte values. In our current proposal,

the addition of the tweak between each round can be seen as a modification of the original AES key schedule, where each subkey after derivation would be XORed with the tweak T. This only translates all the subkeys by a known and fixed constant T, so an adversary cannot take advantage of it. Consequently, the same attacks existing for AES-128 applies to KIASU-BC.

4.3 Security Against Other Attacks on KIASU-BC

As we keep the original round function and key schedule of AES, we believe that the security level of KIASU-BC against the remaining types of attacks stays the same. For instance, the constants in the key schedule stop potential attacks based on similarity of rounds (slide attacks [8]) or on similarity of states (rotational attacks [23], internal differential attacks [31]). The impossible differential attacks on AES [26, 27] do not exploit the key schedule, neither do the truncated differential attacks, and therefore the number of rounds that they can penetrate would remain the same.

A possible increment in the number of attacked rounds might happen in the framework of open-key distinguishers (even though we have not been able to improve the known attacks [11, 17, 20] using this extra tweak input). However, we emphasize that we do not claim any resistance of KIASU-BC in this attack model.

Chapter 5

Software Performances

As KIASU is based on the AES, it allows very efficient software implementation on the processors that support AES-NI. In addition, the mode allows complete parallelization of the AES-NI calls. The actual overhead compared to AES comes only from the tweak schedule. However, this translates only into a single additional XOR of the tweak per round (in terms of AES-NI, if one AES round is implemented as `state=aesenc(state,subkey[i])`, then in KIASU one round can be implemented as `state=aesenc(state,xor(subkey[i],tweak))`). Thus the proposed tweak schedule in KIASU is the simplest possible candidate of the TWEAKEY framework [21].

We used the two Intel processor families Intel Sandy Bridge and Intel Haswell with AES-NI enabled to obtain benchmarks for KIASU. The C implementation of the designs 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 and with no associated data (when the length of the associated data is non-zero, the speed of KIASU= actually increases, while for KIASU≠ stays the same).

Processor	128B	256B	512B	1024B	2048B	4096B
Intel Sandy Bridge	1.41	1.21	1.11	1.06	1.03	1.02
Intel Haswell	0.97	0.84	0.78	0.76	0.75	0.74

Table 5.1: Benchmarks for KIASU≠ expressed in cycles per byte on AES-NI enabled Intel processors.

Processor	128B	256B	512B	1024B	2048B	4096B
Intel Sandy Bridge	5.42	2.93	2.45	2.27	2.07	1.98
Intel Haswell	1.81	1.59	1.48	1.44	1.40	1.39

Table 5.2: Benchmarks for KIASU= expressed in cycles per byte on AES-NI enabled Intel processors.

The results of the benchmarks of the encryption and authentication (simultaneously) for various messages size inputs to KIASU are given in Tables 5.1 and 5.2. The nonce-misuse resistant design KIASU= is roughly two times slower due to the two tweakable block cipher calls per message block (but would get similar speed than KIASU≠ to handle associated data since only a single call per block is required).

The speed of our nonce-respecting design KIASU≠ on both platforms is very close to the speed of OCB3 and outperforms AES-GCM (recall that KIASU≠ provides full 128-bit security!). Another important feature of this design is that it performs quite well on short messages – this fact is due to the chosen mode that does not require additional preprocessing step, like it is the case for OCB3 for example.

We note that KIASU will also perform very well on slightly older or legacy architecture since AES is known to be quite efficient in most situations. KIASU might mark even better than OCB3 since one does not have to implement Galois field multiplication in KIASU. The overhead of the tweak schedule in this type of implementations will be extremely small.

Chapter 6

Hardware Performances

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

Our starting point is the best ASIC lightweight implementation [29] of AES, that only requires 2400GE (from which 70% comes from the memory to store the key and the internal state). Since KIASU-BC is exactly the AES, plus only the tweak layer, it is pretty straightforward to estimate that the overhead will be due to storing the 64-bit tweak value and XORing it to the internal state. This should require around $64 \times (4.67 + 2.67) = 470$ GE, 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. Therefore, we estimate that the entire KIASU-BC can be implemented with around 2900 GE.

Concerning the authenticated encryption mode for KIASU$\neq$, one can remark that it calls directly KIASU-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 KIASU-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 $max_l = 29$ bits, hence $29 \times 4 = 116$ 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 KIASU$\neq$ (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 KIASU-BC calls to the checksum register). Therefore, we do not add extra cost for this part.

As a result, we estimate that KIASU should be able to fit in about 4000 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 KIASU 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 KIASU=, 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 7

Features

The main idea heavily exploited in our designs is the introduction of the the first tweakable block cipher KIASU-BC which is based entirely on AES, but does not use any field multiplications (rather, we only add the tweak value to the state in each round of AES) and which is resistant to all known attacks. We end up with a tweakable primitive that has: 1) close to AES efficiency, and 2) provides full 128-bit security. The most important features of our designs are based precisely on the above two properties as well as on the fact that the authenticated encryption modes we chose allow full parallelization of the cipher calls. Having a fully secure tweakable block cipher allows to obtain very easily a full security authenticated encryption scheme, while modes using AES as a black box usually ensure only birthday security (or are very slow). We provide below a brief overview of these features of KIASU, while keeping as a point of reference AES-GCM [28] and OCB3 [24] – arguably the two most popular authenticated encryption schemes.

- KIASU supports very high security level. In the framework of nonce-respecting adversaries, both AES-GCM and OCB3 provide only 64-bit security, while KIASU$\neq$ has full 128-bit security. Such a feature is extremely important as nowadays computing power makes 64-bit security rather worrying. Moreover, KIASU= was designed for misuse-resistant applications (neither OCB3 nor AES-GCM offers any), and has a provable security of 64 bits. The internally used tweakable block cipher has a security of 128 bits, therefore we conjecture that the actual security level of KIASU= regarding confidentiality and authenticity in nonce-respecting scenario is 128 bits (and thus it would immediately have significant advantage over COPA [2] instantiated with AES).

- KIASU is highly competitive in speed. This comes from the fully parallelizable modes, but also from the use of AES. On processors with AES-NI support, KIASU$\neq$ runs at almost the same speed as OCB3, and much faster than AES-GCM. As it uses only the basic AES operations (including the XOR of the tweak), its speed is comparable to the speed of AES on any other platform (unlike the designs that rely on a special instructions for field multiplications). Therefore, we expect good performances even on legacy platforms since AES is efficient on a broad spectrum of architectures.

- The security of KIASU relies on the security of the proven modes but also on the careful analysis of the tweakable block cipher KIASU-BC. We have shown that the construction resists all potential attacks introduced by the XOR of the tweak and thus as long as AES is secure, KIASU-BC is very likely to stay secure as well.

- The tweakable cipher is interesting on its own and can be used to instantiate other provably secure constructions. It is very easy to implement it – an AES implementation requires only a trivial alteration to result in KIASU-BC. As such, it can benefit from advances regarding side-channels resistant AES implementations. AES-NI implementations would also be resistant to cache-timing attacks [3]. Moreover, it is backwards compatible with AES– KIASU-BC instantiated with a tweak value of 0, results in the original AES. The overhead to implement KIASU on top of AES is therefore minimal.

- KIASU is quite lightweight. This is due to the fact that AES itself can achieve good hardware performances, but also because our tweak schedule is very hardware friendly. In addition, the authenticated encryption modes we use are very lightweight when instantiated with a tweakable block cipher, since only a checksum is required as extra memory for KIASU$\neq$ (two checksums for KIASU=).

- KIASU is extremely fast not only for long messages, but also 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 OCB3 or AES-GCM). The first 128-bit message block is handled directly, and taking in account the tag generation one needs $m+1$ KIASU-BC 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).

- Simplicity was one of our design goal, and KIASU-BC is very straightforward to implement. This is also true for our modes, since using a tweakable block cipher greatly simplifies the authenticated encryption mode definition (no need for field multiplication, generation of masks, etc.). This simplicity will also help the cryptanalysts to study our design.

Chapter 8

Design Rationale

The starting point of our design was to provide the first ad-hoc tweakable version of the AES block cipher (in order to benefit from trust in AES security, and from existing implementation advantages such as AES-NI). Such a primitive is very attractive for many authenticated encryption modes that are secure beyond the birthday bound, but loose this feature when instantiated with a construction that uses AES as a black box (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 because it was believed that tweaking AES is not an easy task and would necessarily lead to a slow design (adding some extra freedom to the attacker seems to enable more powerful attacks and thus implies many more rounds).

Yet, we discovered that using a new type of AES-like tweakable block cipher based on the addition of tweaks to each round of the cipher (the TWEAKEY framework [21]), similar to how the subkeys are added, leads to a very efficient primitive. The construction we have is arguably the simplest possible among all such designs as there is no tweak scheduling algorithm (the same tweak being simply XORed to the state in each round).

From the security analysis performed during the design process of KIASU, we have discovered that this strategy actually limits the tweak size to 64 bits. Indeed, for any larger tweaks the resulting tweakable block cipher is insecure cipher in the related-key related-tweak framework: for 96- or 128-bit tweaks, there exists very high probability related-key related-tweak differentials characteristics, and some are iterative. Building a secure tweakable block cipher that resists such good differential characteristics would therefore require a huge number of rounds. On the other hand, choosing a 64-bit tweak allows a secure construction in the related-key related-tweak model, provided that the 64 bits of the tweak are XORed on the top two rows on the internal state during the AES encryption process. We remark that not all positions to add 64-bit tweaks are secure. For instance, XORing the tweak to the first two columns (rather than rows), results in a tweakable block cipher that is almost trivially insecure in the related-key related-tweak model.

We believe our proposal combine the five most important features: extremely fast in software, good in hardware, full 128-bit security, efficient for both small and long messages, and the possibility to simply switch to a nonce-misuse mode if needed.

The designer/designers have not hidden any weaknesses in this cipher.

Chapter 9

Intellectual Property

KIASU 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 KIASU uses a mode belonging to the generic ΘCB3 framework, it is unclear if patents relative to OCB3 (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.

If any of this information changes, the submitter will promptly (and within at most one month) announce these changes on the crypto-competitions mailing list.

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] : Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), FIPS PUB 197, U.S. Department of Commerce (November 2001)

[2] 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

[3] Bernstein, D.J.: Cache-timing attacks on AES. http://cr.yp.to/antiforgery/cachetiming-20050414.pdf (2005)

[4] 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

[5] 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

[6] Biryukov, A., Nikolic, 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 2010. Volume 6110 of LNCS., Springer (May 2010) 322–344

[7] Biryukov, A., Nikolic, I.: Search for Related-Key Differential Characteristics in DES-Like Ciphers. [22] 18–34

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

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

[11] 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

[12] 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

[13] 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

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

[15] 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

[16] 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

[17] Gilbert, H., Peyrin, T.: Super-Sbox Cryptanalysis: Improved Attacks for AES-Like Permutations. [19] 365–383

[18] Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In Preneel, B., Takagi, T., eds.: CHES 2011. Volume 6917 of LNCS., Springer (September / October 2011) 326–341

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

[20] 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

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

[22] Joux, A., ed.: FSE 2011. In Joux, A., ed.: FSE 2011. Volume 6733 of LNCS., Springer (February 2011)

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

[24] Krovetz, T., Rogaway, P.: The Software Performance of Authenticated-Encryption Modes. [22] 306–327

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

[26] Lu, J., Dunkelman, O., Keller, N., Kim, J.: New Impossible Differential Attacks on AES. In Chowdhury, D.R., Rijmen, V., Das, A., eds.: INDOCRYPT 2008. Volume 5365 of LNCS., Springer (December 2008) 279–293

[27] Mala, H., Dakhilalian, M., Rijmen, V., Modarres-Hashemi, M.: Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In Gong, G., Gupta, K.C., eds.: INDOCRYPT 2010. Volume 6498 of LNCS., Springer (December 2010) 282–291

[28] 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)

[29] 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

[30] 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

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

[32] 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

[33] 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 $\mathcal{S}(0x25) = 0x3F$.

	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 $\mathcal{S}(0x3F) = 0x25$.

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.

8d	01	02	04	08	10	20	40	80	1b	36	6c	d8	ab	4d	9a
2f	5e	bc	63	c6	97	35	6a	d4	b3	7d	fa	ef	c5	91	39
72	e4	d3	bd	61	c2	9f	25	4a	94	33	66	cc	83	1d	3a
74	e8	cb	8d	01	02	04	08	10	20	40	80	1b	36	6c	d8
ab	4d	9a	2f	5e	bc	63	c6	97	35	6a	d4	b3	7d	fa	ef
c5	91	39	72	e4	d3	bd	61	c2	9f	25	4a	94	33	66	cc
83	1d	3a	74	e8	cb	8d	01	02	04	08	10	20	40	80	1b
36	6c	d8	ab	4d	9a	2f	5e	bc	63	c6	97	35	6a	d4	b3
7d	fa	ef	c5	91	39	72	e4	d3	bd	61	c2	9f	25	4a	94
33	66	cc	83	1d	3a	74	e8	cb	8d	01	02	04	08	10	20
40	80	1b	36	6c	d8	ab	4d	9a	2f	5e	bc	63	c6	97	35
6a	d4	b3	7d	fa	ef	c5	91	39	72	e4	d3	bd	61	c2	9f
25	4a	94	33	66	cc	83	1d	3a	74	e8	cb	8d	01	02	04
08	10	20	40	80	1b	36	6c	d8	ab	4d	9a	2f	5e	bc	63
c6	97	35	6a	d4	b3	7d	fa	ef	c5	91	39	72	e4	d3	bd
61	c2	9f	25	4a	94	33	66	cc	83	1d	3a	74	e8	cb	8d

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] = 1, RCON(2) = 2 and RCON(9)=0x1b.