Joltik v1.3

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

http://www1.spms.ntu.edu.sg/~syllab/Joltik

August 28, 2015

Chapter 1

Introduction

In this note, we propose Joltik, a new authenticated encryption design based on a 64-bit lightweight tweakable block cipher Joltik-BC that uses an AES-like 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 [21]) allowing designers to unify the vision of key and tweak inputs of a cipher. We plug this cipher into two different fully parallel and provably secure authenticated encryption modes: one for which the nonce must not be reused, the other one providing security even when the nonce is reused.

In short, Joltik is a lightweight authenticated encryption scheme oriented especially for hardware applications. It can be implemented with low area footprint (Joltik-BC is a lightweight cipher and some extra area is required for the memory in the authenticated encryption mode). It performs very well for small messages (only $m + 1$ calls to Joltik-BC are required for a m block message and without any precomputation), in contrary to sponge or stream cipher based lightweight designs that require a strong initialization stage. Such a feature is very important as many constrained environments will only cipher very short messages (for example a 96-bit Electronic Product Code). In some versions of Joltik, the key can be hard-wired for even smaller implementations when the application permits. The overhead for decryption capability is minimal, as we use a novel very lightweight MDS diffusion matrix which is involutory. Joltik provides a 64-bit security for both privacy and authenticity, similarly to AES-GCM [31] or OCB [25] which only ensure a birthday bound security on 128 bits. Even being hardware-oriented, Joltik performs rather well on software platforms, using recent bitsliced implementations of AES-like lightweight ciphers [1,30]. In the nonce-misuse resistant versions of Joltik, in addition to a full 64-bit security for unique nonces, we obtain birthday-bound security (not an online nonce-misuse resistance as defined in [15], but a full MRAE security notion [37]) when the nonce is reused. This is done very simply as a tweakable block cipher is a quite handy primitive to build an authenticated encryption scheme.

Organization of the document. In Chapter 2, we provide the specification of our proposal Joltik, 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 provide a security analysis of the proposal. In Chapters 5 and 6, we detail the design decisions, and in Chapter 7 we estimate the hardware performances of the various versions of Joltik. We finish with Chapters 8 and 9 where we give notes on intellectual property and consent.

Chapter 2

Specification

In this chapter, we present a full specification of our proposal `Joltik`. We first give the recommended parameter sets and then proceed with the description of the design. We explain the two authenticated encryption modes $\texttt{Joltik}^{\neq}$ and $\texttt{Joltik}^{=}$, and then we describe the lightweight ad-hoc tweakable block cipher `Joltik-BC` (which is based on the `TWEAKEY` framework [21]) 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 `Joltik-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 $|X| < n$. For an empty string ϵ, the 10^* padding will not add any bit: $pad10^*(\epsilon) = \epsilon$. The truncation of the word X to the first i bits is given by $\lceil X \rceil_i$, and the truncation to the last i bits by $\lfloor X \rfloor_i$. Moreover, $X \lll a$ will denote the word X rotated by a positions to the left.

Our authenticated encryption scheme `Joltik` 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 (the same limitation applies to the associated data material). We have that $max_l = 2^{\lceil t/2 \rceil - 4}$ 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 $|N| \geq \log_2(max_m) = \lfloor t/2 \rfloor$. Note that there is a tradeoff possible here between max_l and max_m, as long as $max_l \cdot max_m = 2^{t-4}$.

2.1 Parameters

A first parameter for `Joltik` is the key length k, which is between 64 and 128 bits. We then propose two modes: the first is for nonce-respecting adversaries (denoted with a $\neq$ sign), while the second offers nonce misuse-resistance (denoted with a $=$ sign). For this reason, we introduce another parameter, that signals the mode of our authenticated encryption scheme. The tag size τ is recommended to be 64 bits, while the public message length $|N|$ is between 32 bits and 56 bits for the first mode, and 64 bits for the second mode.

2.2 Recommended Parameter Sets

The public message number is the nonce. For each of the two modes we recommend four parameter sets (hence in total, we have 8 sets), listed in Table 2.1. The list is sorted from most important to least important. We denote by $\texttt{Joltik}^{\neq}$ the design in the case of the nonce-respecting mode and $\texttt{Joltik}^{=}$ the design in the case of the nonce-misuse resistant mode.

Name	k	t	n	$\lvert N\rvert$	τ
$\texttt{Joltik}^{\neq}$-64-64	64	64	64	32	64
$\texttt{Joltik}^{\neq}$-80-112	80	112	64	56	64
$\texttt{Joltik}^{\neq}$-96-96	96	96	64	48	64
$\texttt{Joltik}^{\neq}$-128-64	128	64	64	32	64
$\texttt{Joltik}^{=}$-64-64	64	64	64	64	64
$\texttt{Joltik}^{=}$-80-112	80	112	64	64	64
$\texttt{Joltik}^{=}$-96-96	96	96	64	64	64
$\texttt{Joltik}^{=}$-128-64	128	64	64	64	64

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

$\texttt{Joltik}^{\neq}$-64-64 and $\texttt{Joltik}^{=}$-64-64 are based on the internal block cipher Joltik-BC-128, while $\texttt{Joltik}^{\neq}$-80-112, $\texttt{Joltik}^{=}$-80-112, $\texttt{Joltik}^{\neq}$-96-96, $\texttt{Joltik}^{=}$-96-96, $\texttt{Joltik}^{\neq}$-128-64, and $\texttt{Joltik}^{=}$-128-64 are based on the internal block cipher Joltik-BC-192.

2.3 The Authenticated Encryption Joltik

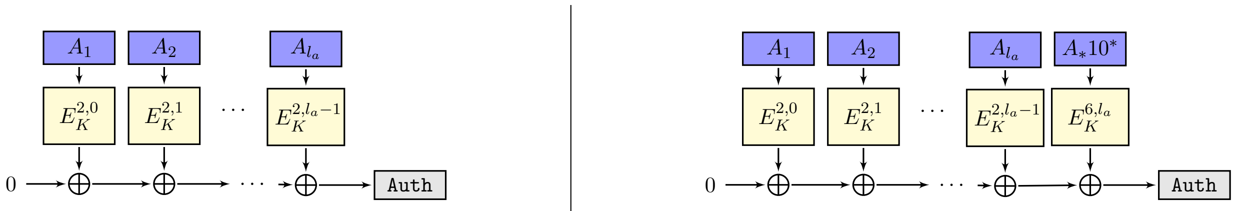
In this section, we provide the high-level description of our proposal. Joltik uses a tweakable block cipher Joltik-BC as internal primitive (specified in Section 2.4), and we describe here the simple authenticated encryption modes built on top of it. Joltik has two main mode variants:

- $\mathcal{E}^{\neq}$ 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 similar to TAE [26, 27] or ΘCB3 [25] (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, a new authenticated encryption mode named SCT [35], 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).

In both modes, we use short 4-bit prefixes for the tweak input in order to properly separate the various types of encryption/authentication blocks. Is is to be noted that the two modes are actually quite similar, the main difference being that the first one applies one pass on the message blocks, while the second performs two passes (which is necessary to obtain a MRAE security notion).

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 [25] 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: in the case where the associated data is a multiple of the block size, no padding is needed.

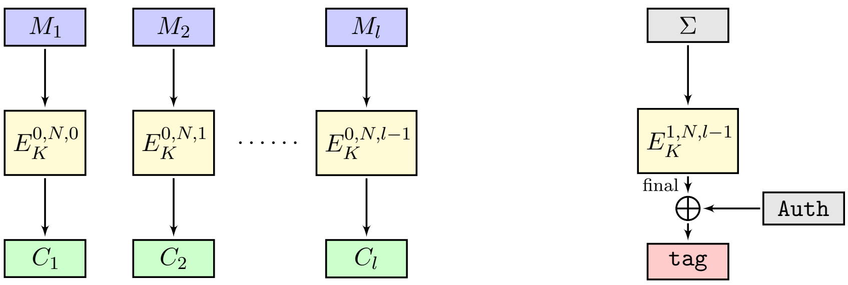


Figure 2.2: Message processing for the nonce-respecting mode: in the case where the message-length is a multiple of the block size, no padding is needed.

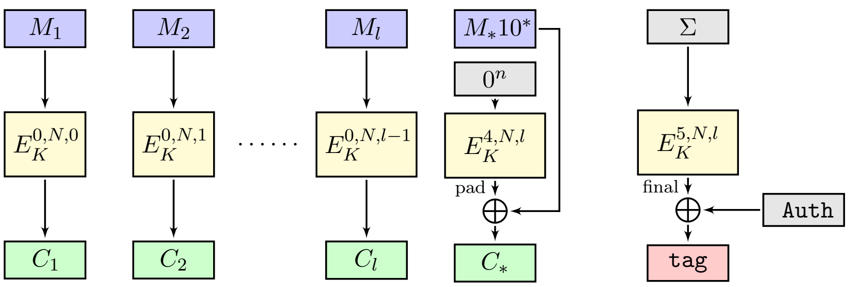


Figure 2.3: Message processing for the nonce-respecting mode: in the case where the message-length is a not multiple of the block size, padding is needed. Note that the checksum Σ is computed with a 10^* padding for block M^*.

Algorithm 1: The encryption algorithm $\mathcal{E}_K^{\neq}(N, A, M)$.
In the tweak inputs, the value N is encoded on $\log_2(max_m)$ bits, the integer values j and l are encoded on $\log_2(max_l)$ bits, while the integer values i and l_a are encoded on $\log_2(max_l \cdot max_m) = t - 4$ bits.

```
/* Associated data */
A_1||...||A_{l_a}||A_* ← A where each |A_i| = n and |A_*| < n
Auth ← 0
for i = 0 to l_a − 1 do
  | Auth ← Auth ⊕ E_K(0010||i, A_{i+1})
end
if A_* ≠ ϵ then
  | Auth ← Auth ⊕ E_K(0110||l_a, pad10*(A_*))
end

/* Message */
M_1||...||M_l||M_* ← M where each |M_j| = n and |M_*| < n
Checksum ← 0^n
for j = 0 to l − 1 do
  | Checksum ← Checksum ⊕ M_j
  | C_j ← E_K(0000||N||j, M_{j+1})
end
if M_* = ϵ then
  | Final ← E_K(0001||N||l − 1, Checksum)
  | C_* ← ϵ
else
  | Checksum ← Checksum ⊕ pad10*(M_*)
  | Pad ← E_K(0100||N||l, 0^n)
  | C_* ← M_* ⊕ ⌈Pad⌉_{|M_*|}
  | Final ← E_K(0101||N||l, Checksum)
end

/* Tag generation */
tag ← Final ⊕ Auth
return (C_1||...||C_l||C_*, tag)
```

Algorithm 2: The verification/decryption algorithm $\mathcal{D}^{\neq}_K(N, A, C, \texttt{tag})$.
In the tweak inputs, the value N is encoded on $\log_2(max_m)$ bits, the integer values j and l are encoded on $\log_2(max_l)$ bits, while the integer values i and l_a are encoded on $\log_2(max_l \cdot max_m) = t - 4$ bits.

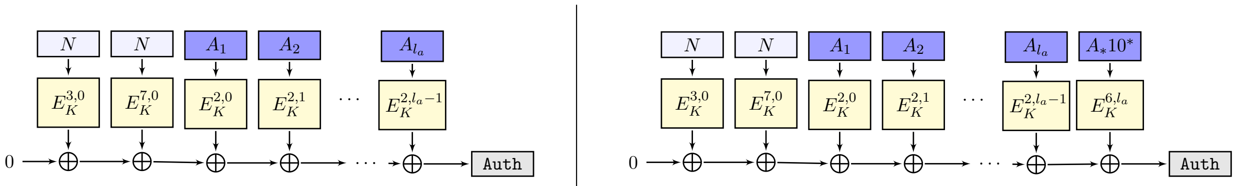
/* *Associated data* */
$A_1||\ldots||A_{l_a}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$
$\text{Auth} \leftarrow 0$
for $i = 0$ **to** $l_a - 1$ **do**
    $\text{Auth} \leftarrow \text{Auth} \oplus E_K(0010||i, A_{i+1})$
end
if $A_* \neq \epsilon$ **then**
    $\text{Auth} \leftarrow \text{Auth} \oplus E_K(0110||l_a, pad10^*(A_*))$
end

/* *Ciphertext* */
$C_1||\ldots||C_l||C_* \leftarrow C$ where each $|C_j| = n$ and $|C_*| < n$
$\text{Checksum} \leftarrow 0^n$
for $j = 0$ **to** $l - 1$ **do**
    $M_j \leftarrow D_K(0000||N||j, C_{j+1})$
    $\text{Checksum} \leftarrow \text{Checksum} \oplus M_j$
end
if $C_* = \epsilon$ **then**
    $\text{Final} \leftarrow E_K(0001||N||l - 1, \text{Checksum})$
    $M_* \leftarrow \epsilon$
else
    $\text{Pad} \leftarrow E_K(0100||N||l, 0^n)$
    $M_* \leftarrow C_* \oplus \lceil \text{Pad} \rceil_{|C_*|}$
    $\text{Checksum} \leftarrow \text{Checksum} \oplus pad10^*(M_*)$
    $\text{Final} \leftarrow E_K(0101||N||l, \text{Checksum})$
end

/* *Tag verification* */
$\texttt{tag}' \leftarrow \text{Final} \oplus \text{Auth}$
if $\texttt{tag}' = \texttt{tag}$ **then return** $(M_1||\ldots||M_l||M_*)$
else return $\perp$

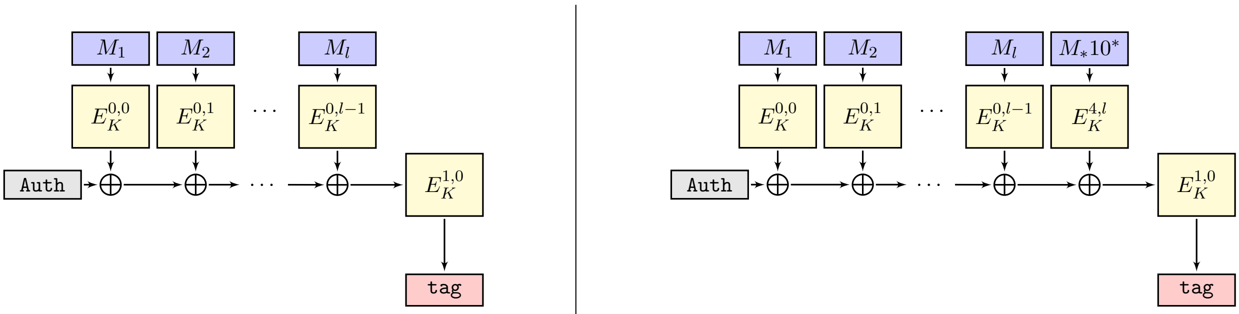
2.3.2 Nonce-Misuse Resistant Mode: $\mathcal{E}^{=}$ and $\mathcal{D}^{=}$

The encryption algorithm $\mathcal{E}^{=}$ is depicted in Figures 2.4 and 2.5 for the authentication part and in Figure 2.6 for the encryption part. An algorithmic description is given in Algorithm 3. The verification/decryption algorithmic description of $\mathcal{D}^{=}$ is given in Algorithm 4. Our mode is a new authenticated encryption mode, named Synthetic Counter in Tweak (SCT), that uses a tweakable block cipher as internal primitive [35].



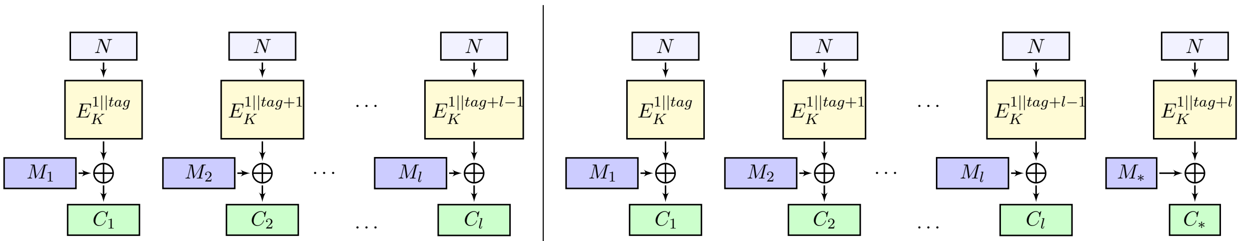
(a) Without padding. (b) With padding.

Figure 2.4: Handling of the associated data for the nonce-misuse resisting mode: in the case where the associated data is a multiple of the block size, no padding is needed.



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

Figure 2.5: Message processing in the authentication part of the nonce-misuse resisting mode: in the case where the message-length is a multiple of the block size, no padding is needed.



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

Figure 2.6: Message processing for the encryption part of the nonce-misuse resisting mode.

Algorithm 3: The encryption algorithm $\mathcal{E}_K^{\equiv}(N, A, M)$.
In the tweak inputs, the integer values i, j, l and l_a are encoded on $\log_2(max_l \cdot max_m) = t - 4$ bits. Moreover, the $tag + j$ values are encoded on $t - 1$ bits (the most significant bit is truncated since $|\texttt{tag}| = t$).

```
/* Associated data */
A_1||...||A_{l_a}||A_* ← A where each |A_i| = n and |A_*| < n
Auth ← E_K(0011||0^{t−4}, N) ⊕ E_K(0111||0^{t−4}, N)
for i = 0 to l_a − 1 do
  | Auth ← Auth ⊕ E_K(0010||i, A_{i+1})
end
if A_* ≠ ϵ then
  | Auth ← Auth ⊕ E_K(0110||l_a, pad10*(A_*))
end

/* Message authentication and tag generation */
M_1||...||M_l||M_* ← M where each |M_j| = n and |M_*| < n
tag ← Auth
for j = 0 to l − 1 do
  | tag ← tag ⊕ E_K(0000||j, M_{j+1})
end
if M_* ≠ ϵ then
  | tag ← tag ⊕ E_K(0100||l, pad10*(M_*))
end
tag ← E_K(0001||0^{t−4}, tag)

/* Message encryption */
for j = 0 to l − 1 do
  | C_j ← M_j ⊕ E_K(1||tag + j, N)
end
if M_* ≠ ϵ then
  | C_* ← M_* ⊕ E_K(1||tag + l, N)
end

return (C_1||...||C_l||C_*, tag)
```

Algorithm 4: The verification/decryption algorithm $\mathcal{D}_K^{\equiv}(N, A, C, \texttt{tag})$.
In the tweak inputs, the integer values i, j, l and l_a are encoded on $\log_2(max_l \cdot max_m) = t - 4$ bits. Moreover, the $tag + j$ values are encoded on $t - 1$ bits (the most significant bit is truncated since $|\texttt{tag}| = t$).

/ Message decryption */*
$C_1||\ldots||C_l||C_* \leftarrow C$ where each $|C_j| = n$ and $|C_*| < n$
for $j = 0$ **to** $l - 1$ **do**
  $M_j \leftarrow C_j \oplus E_K(1||\texttt{tag} + j, N)$
end
if $C_* \neq \epsilon$ **then**
  $M_* \leftarrow C_* \oplus E_K(1||\texttt{tag} + l, N)$
end

/ Associated data */*
$A_1||\ldots||A_{l_a}||A_* \leftarrow A$ where each $|A_i| = n$ and $|A_*| < n$
$\text{Auth} \leftarrow E_K(0011||0^{t-4}, N) \oplus E_K(0111||0^{t-4}, N)$
for $i = 0$ **to** $l_a - 1$ **do**
  $\text{Auth} \leftarrow \text{Auth} \oplus E_K(0010||i, A_{i+1})$
end
if $A_* \neq \epsilon$ **then**
  $\text{Auth} \leftarrow \text{Auth} \oplus E_K(0110||l_a, pad10^*(A_*))$
end

/ Message authentication and tag generation */*
$M_1||\ldots||M_l||M_* \leftarrow M$ where each $|M_j| = n$ and $|M_*| < n$
$\texttt{tag}' \leftarrow \text{Auth}$
for $j = 0$ **to** $l - 1$ **do**
  $\texttt{tag}' \leftarrow \texttt{tag}' \oplus E_K(0000||j, M_{j+1})$
end
if $M_* \neq \epsilon$ **then**
  $\texttt{tag}' \leftarrow \texttt{tag} \oplus E_K(0100||l, pad10^*(M_*))$
end
$\texttt{tag}' \leftarrow E_K(0001||0^{t-4}, \texttt{tag}')$

/ Tag verification */*
if $\texttt{tag}' = \texttt{tag}$ **then return** $(M_1||\ldots||M_l||M_*)$
else return $\perp$

2.4 The Tweakable Block Cipher Joltik-BC

Joltik-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 64-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, Joltik-BC-128 for which the cumulative size of the key and the tweak is 128 bits, and Joltik-BC-192 for which the cumulative size of the key and the tweak is 192 bits.

Joltik-BC is an AES-like design, i.e. it is an iterative substitution-permutation network 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 Joltik-BC is seen as 4×4 matrix of nibbles, where each nibble is a 4-bit word (we denote the nibble size with c, thus $c = 4$). We denote the base field by $\mathbb{K}$ as $GF(16)$ defined by the irreducible polynomial $x^4 + x + 1$ (sometimes noted in hexadecimal display 0x13). The number r of rounds is 24 for Joltik-BC-128 and 32 for Joltik-BC-192. 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 64-bit round subtweakey (defined further) to the internal state,
- SubNibbles – Apply the 4-bit S-Box $\mathcal{S}$ defined below to the 16 nibbles of the internal state,
- 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 $\mathbf{M}$ defined below.

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

The 4-bit S-Box $\mathcal{S}$ we use in Joltik-BC is the one selected for the Piccolo block cipher [38], and is exhaustively defined by:

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

The MDS matrix $\mathbf{M}$ with coefficients in $\mathbb{K}$ we use in Joltik-BC is non-circulant, MDS and involutory:

$$\mathbf{M} = \begin{pmatrix} 1 & 4 & 9 & 13 \\ 4 & 1 & 13 & 9 \\ 9 & 13 & 1 & 4 \\ 13 & 9 & 4 & 1 \end{pmatrix} = \mathbf{M}^{-1}.$$

We refer to [22] for more details on this class of matrices.

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 64-bit round subtweakey to the internal state,
- invMixNibbles – Multiply the internal state by a 4×4 MDS matrix $\mathbf{M}^{-1}$ with coefficients in $\mathbb{K}$ previously defined (we recall that $\mathbf{M} = \mathbf{M}^{-1}$ as $\mathbf{M}$ is an involution),
- 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 produce the plaintext value.

Definition of the subtweakeys. The description of the cipher above has so far followed the classical construction of an AES-like block cipher. The operation AddRoundTweakey, and in particular the production of the subtweakeys, is where Joltik-BC differs from the other ciphers.

Let us denote with STK_i the subtweakey (a 64-bit word) that is added to the state at round i of the cipher with the AddRoundTweakey operation. For Joltik-BC-128, subtweakey is defined as:

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

whereas for Joltik-BC-192 is defined as:

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

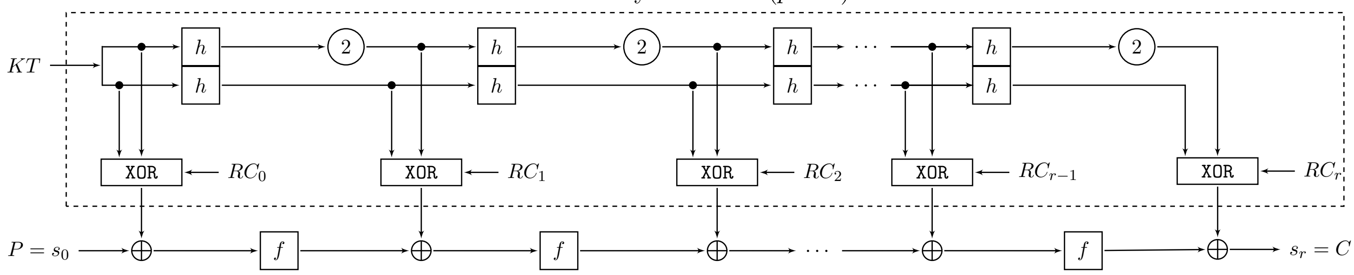


Figure 2.7: Instantiation of the TWEAKEY framework for Joltik-BC-128.

The 64-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 64-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 nibble permutation h, and a finite field multiplication g:

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

The nibble 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 a finite field multiplication in $\mathbb{K}$ 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 Joltik-BC-128, the size of KT is 128 bits. The first (most significant) 64 bits of KT is W_1, while the second W_2. Then, TK_i^1 are the output words of the key scheduling algorithm $KS(W_1, 1)$, and TK_i^2 are the output words of the key scheduling algorithm $KS(W_2, 2)$. For Joltik-BC-192, the size of KT is 192 bits, there are three words W_1, W_2, W_3, and TK_i^1 are outputs of $KS(W_1, 1)$, TK_i^2 are the outputs of $KS(W_2, 2)$, and TK_i^3 are outputs of $KS(W_3, 4)$. We note that the second inputs to the KS functions are different and such that the first key word can be hard-wired if needed (since the bytes of this word will be multiplied by coefficient 1, thus not multiplied).

Finally, RC_i are the key schedule round constants (similar to the constants used in `LED` cipher [18]), and are defined as:

$$RC_i = \begin{pmatrix} 0 & (rc_5||rc_4||rc_3) & 0 & 0 \\ 1 & (rc_2||rc_1||rc_0) & 0 & 0 \\ 2 & (rc_5||rc_4||rc_3) & 0 & 0 \\ 3 & (rc_2||rc_1||rc_0) & 0 & 0 \end{pmatrix}.$$

The values of $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ are given in Appendix A.2.

Chapter 3

Security Claims

We provide our security claims for the different variants of `Joltik` 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 below the security goals expressed in terms of k and the block size n.

One can see that we do claim full k-bit security for both $\texttt{Joltik}^{\neq}$ and $\texttt{Joltik}^{=}$ for a nonce-respecting user, in contrary to other modes like `AES-GCM` [31] or `OCB3` [25], which only ensure birthday-bound security. In the nonce-misuse scenario, we claim a birthday-bound security concerning $\texttt{Joltik}^{=}$.

	Security (bits)	
Goal (nonce-respecting user)	$\texttt{Joltik}^{\neq}$	$\texttt{Joltik}^{=}$
key recovery	k	k
Confidentiality for the plaintext	n	$n-1$
Integrity for the plaintext	n	$n-1$
Integrity for the associated data	n	$n-1$
Integrity for the public message number	n	$n-1$

	Security (bits)	
Goal (nonce-misuse user)	$\texttt{Joltik}^{\neq}$	$\texttt{Joltik}^{=}$
key recovery	k	k
Confidentiality for the plaintext	none	$n/2$
Integrity for the plaintext	none	$n/2$
Integrity for the associated data	none	$n/2$
Integrity for the public message number	none	$n/2$

Table 3.1: Security goals of `Joltik`. 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 (nonce-misuse user). The bit security of our designs is expressed in terms of calls to the internal tweakable block cipher, up to a small logarithmic factor.

In the table we assume the public message number is a nonce and there is no secret message number. We also assume that for the nonce-respecting mode, the total size of the associated data and the total size of the message do not exceed $8 \cdot 2^{max_l}$ bytes. Thus, this amounts to 2^{31} bytes for $\texttt{Joltik}^{\neq}\texttt{-64-64}$ and $\texttt{Joltik}^{\neq}\texttt{-128-64}$, 2^{47} bytes for $\texttt{Joltik}^{\neq}\texttt{-96-96}$, and 2^{55} bytes for $\texttt{Joltik}^{\neq}\texttt{-80-112}$. For the nonce-misuse resistant mode, the total size of the associated data and the total size of the message do not exceed $16 \cdot 2^{max_l} \cdot 2^{max_m}$ bytes, thus 2^{63}

bytes for $\texttt{Joltik}^{=}$-64-64 and $\texttt{Joltik}^{=}$-128-64, 2^{95} bytes for $\texttt{Joltik}^{=}$-96-96, and 2^{111} bytes for $\texttt{Joltik}^{=}$-80-112. Moreover, the maximum number of messages that can be handled for a same key is 2^{max_m}, that is 2^{32} for $\texttt{Joltik}^{\neq}$-128-64, $\texttt{Joltik}^{\neq}$-128-64, $\texttt{Joltik}^{=}$-64-64 and $\texttt{Joltik}^{=}$-128-64, 2^{48} for $\texttt{Joltik}^{\neq}$-96-96 and $\texttt{Joltik}^{=}$-96-96, and 2^{56} for $\texttt{Joltik}^{\neq}$-80-112 and $\texttt{Joltik}^{=}$-80-112.

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

Chapter 4

Security Analysis

Our design can be seen as an instantiation of secure authenticated encryption modes based on a tweakable block cipher. However, the security goal of this cipher, unlike most of the tweakable block ciphers that only ensure birthday bound security when plugged in the mode, is much higher – indeed we claim full 64-bit security against all possible attacks in the secret key model. Our claims are based on the AES wide-trail strategy which trivially provides resistance against the classical differential and linear cryptanalysis in the single key model and on the fact that we can search for all related-key/tweak differential trails in the related-key model. We study as well the additional threats introduced by the key/tweak schedule against the Meet-in-the-Middle (MITM) attacks.

The TWEAKEY framework [21] allows a dual view of the whole tweakable block cipher 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 (assure 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 [4, 5, 14, 16, 32, 39] that given a key schedule can return the upper bound on probability of the best related-key differential trails, 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 trails, 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 trails. 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 [4], split approach [5], and extended split approach [14]. We omit the details on how these search algorithms operate due to their complexity, and further, give only the final results (refer to Tables 4.1 and 4.2) produced by the tools.

We have to run two completely different searches due to the accumulative size of the subkeys and the subtweaks added in each round of the cipher (this size is 128 bits for Joltik-BC-128 and 192 bits for Joltik-BC-192). The best differential trails for these two variants are different as in the case of the later, the trails are on more rounds (obviously this comes from the fact that the subkeys and subtweaks nibbles can be canceled more times than in the case of Joltik-BC-128).

rounds	active S-Boxes	upper bound on probability	method used
1	0	2^0	trivial
2	0	2^0	trivial
3	1	2^{-2}	Matsui's
4	5	2^{-8}	Matsui's
5	9	2^{-18}	Matsui's
6	12	2^{-24}	Matsui's
8	≥ 17	2^{-34}	extended split (4R+4R)
10	≥ 22	2^{-44}	extended split (5R+5R)
16	≥ 34	2^{-68}	split (8R+8R)

Table 4.1: `Joltik-BC`-128: Upper bounds on probability of the best round-reduced related-key related-tweak differential trails.

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^{-2}	Matsui's
5	4	2^{-8}	Matsui's
6	8	2^{-12}	Matsui's
12	≥ 22	2^{-44}	extended split (6R+6R)
24	≥ 44	2^{-88}	split (12R+12R)

Table 4.2: `Joltik-BC`-192: Upper bounds on probability of the best round-reduced related-key related-tweak differential trails.

The final results of the searches are as follows. For the tweakable block cipher `Joltik-BC`-128, all related-key related-tweak differential trails on 16 or more rounds have upper bound on probability of 2^{-68}. On the other hand, for the tweakable block cipher `Joltik-BC`-192, all related-key related-tweak differential trails on 24 or more rounds have upper bound on probability of 2^{-88}. Both of the bounds are below 2^{-64} (recall that the block size is 64 bits). Note that these are only bounds and we do not have actual trails that match these bounds, hence the actual probabilities might be much lower. This, as well as the fact that each cipher has 8 additional rounds compared to the bound (in case of `Joltik-BC`-128 we have 24 rounds, for `Joltik-BC`-192 we have 32 rounds), gives the ciphers a large security margin against all related-key/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 [4, 5, 14, 16, 32, 39]. Indeed, this attack strongly relies on the `AES` key schedule to propagate linear equations in the meet-in-the-middle strategy to spare some guesses in both the offline and online phases. As the design we propose introduces a new way to handle the key material, it is of interest to see how it interacts with the AES round function.

For a given tweak value, Joltik-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 advanced 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. Interestingly, the SQUARE key schedule, which is also fully linear, is immune (to date) to a meet-in-the-middle attack on its full 8 rounds.

4.3 Security of Other Attacks of Joltik-BC

The slide attack is a block cipher cryptanalysis technique [6] that exploits the degree of self-similarity of a permutation. In Joltik-BC, we have chosen distinct round-dependent constants, which makes the slide attack impossible to perform.

The integral cryptanalysis [9] of AES-based designs can be very efficient, but only works for a small number of rounds. In Joltik-BC, we have picked a large number of rounds (24 or 32 depending on the version) so that we cannot conduct this attack on the full version of the cipher.

Rotational cryptanalysis [23] is another class of attacks, which compares the evolution of a rotated variant of an input words through the encryption process. While this attack has been successfully applied to ARX designs, but cannot be applied to date to byte-oriented ciphers like AES-based ciphers, including Joltik-BC. Moreover, this type of attack, as well as internal differential attack [34], are stopped by the use of constants in the key schedule.

The impossible differential attack on AES [28, 29] does not exploit the key schedule, so the same technique could be applied on Joltik-BC, and achieve the same number of rounds. Again, the security margin of Joltik-BC due to the number of rounds is large enough to resist to an attack on the full primitive.

Since by design, there is no distinction between key and tweak (rather the key + tweak inputs are treated as one tweakey input), trivial so-called related-cipher attacks [40] would apply to two different versions of the Joltik-BC. As the practical threat coming from this type of attack framework is unclear, we decided not to put different constants RC_i in order to prevent the attack.

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, 17, 20] using this extra tweak input). However, we emphasize that we do not claim any resistance of Joltik-BC in this attack model.

Chapter 5

Features

The main idea heavily exploited in the design of `Joltik` is the introduction of a lightweight tweakable block cipher `Joltik-BC`, belonging to the family of the well-known `AES`-like primitives. The tweakable block cipher is a secure instantiation of a more general framework (`TWEAKEY` [21]) and does not rely on big field multiplications as previous tweakable ciphers proposals. Structurally, `Joltik-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.

This design especially targets constrained environments as the hardware footprint is very small, and the authentication encryption scheme built on top of it comes with a very low overhead. Indeed, while we use `OCB3` [25] mode which ensures only birthday-bound security with a classical block cipher, `Joltik` brings full security at a very low extra cost. We detail more precisely below the main features on `Joltik`.

- `Joltik` is very lightweight. We crafted very small 64-bit tweakable block ciphers to obtain a lightweight overall authenticated encryption design. Moreover, when any of our two modes is instantiated with a tweakable block ciphers, the area required is maintained at a low level since no computation in big fields are required anymore in order to build the tweakable block cipher from a classical block cipher. For example, we estimate that $\texttt{Joltik}^{\neq}\texttt{-64-64}$ can be implemented in hardware with around 2100 GE, while $\texttt{Joltik}^{\neq}\texttt{-80-112}$, $\texttt{Joltik}^{\neq}\texttt{-96-96}$ and $\texttt{Joltik}^{\neq}\texttt{-128-64}$ would need around 2600 GE.

- `Joltik` offers full security with only one call to the block cipher per message block on average. In comparison, the two main modes for authenticated encryption to date, namely `OCB3` and `AES-GCM` [31], ensure security up to the birthday bound, so that very few situations in the area of lightweight cryptography would allow to have such a low security with a 64-bit block cipher. Some previous attempts to build full-security, dedicated `AES`-based authenticated encryption schemes have appeared in the past (see `ALE` [7] or `FIDES` [3]), however both have been broken, in [24] and [13] respectively.

- `Joltik` behaves very good 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 64-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).

- `Joltik` has a good security margin for all the recommended parameters. We measure the security margin in terms of number of rounds: the smallest variant of `Joltik` counts 24 rounds, while the best known attacks on `AES`-based design can reach around 7 to 14 rounds, depending of the adversarial model. The largest version of `Joltik` subjects to the same attack offers an even bigger security margin with 32 rounds. Interestingly, `Joltik-BC` is very similar to LED, but we have shown that a good key schedule can significantly reduce the number of rounds required for a secure AES-like cipher: Joltik-BC with 128-bit key/tweak space has only 24 rounds, whereas LED with 128-bit key has 48, and a few attacks (see [12, 33]) reach up to 32-rounds of the cipher.

- The security arguments of Joltik are directly inherited from the two modes used in our design. Indeed, for nonce-respecting users, Joltik benefits from the proofs of the OCB3 mode, while for nonce-repeating users, we designed a new provably secure authenticated encryption mode named SCT [35].

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

- Joltik 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 Joltik is also easy, and can reuse the design strategies, implementations and optimizations available for the AES and lightweight AES-like ciphers [2].

- Joltik-BC is a cipher intended for lightweight, but we can approximate the speed in software. Based on the result from [2] of implementing lightweight block ciphers in software (in particular on the speed of LED), we can give a rough estimate of the speed of Joltik-BC. As the number of rounds in Joltik-BC does not exceed the number of rounds in LED-64, and Joltik-BC has a bit more complex key schedule, we expect that our tweakable block cipher will run in around 15 cycles per byte on the latest Intel processors.

- Joltik 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 Joltik.

- Joltik-BC has smooth parameters handling. We define some recommended parameter sets in this document, but any user can pick its own variant of Joltik-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 accumulative size). This feature extends to the whole Joltik design.

Chapter 6

Design Rationale

The starting point of our design is to provide the first ad-hoc lightweight tweakable block cipher that has a 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 mode that use a block cipher remain quite slow). Therefore, designing a secure tweakable block cipher would enable us to reach full 64-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.

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

6.1 Details for the STK construction

Designing a secure round function for block ciphers has become a fairly easy task – an S-Box layer and a diffusion layer based on MDS code immediately provide good security 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 subtwekeys. 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 $\alpha_i, i = 1, \ldots, p$) 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 maximize the number of nonzero 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 if all $p \times p$ submatrices of $\mathbf{V}$ have maximal rank. We note that this cannot be true for a big number of rounds, since for example the α_i^j values will cycle when the finite field is small (for example in GF(2^4)). However, since the security proofs we will aim only apply to a rather small number of rounds r', we only need to ensure this property for $r' \ll r$ rounds.

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 accumulative difference coming from the subkeys can be canceled only once by XOR of the subkeys (but the remaining will have this difference). For TK-3, this 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 128-bit key and 64-bit block, we were able to obtain a tweakable block cipher Joltik-BC-128 that can be used with 64-bit key and 64-bit tweak. A similar transition was made from the TK-3 block cipher Joltik-BC-192 to 128-bit key and 64-bit tweak, or with 96-bit key and 96-bit tweak, or with 80-bit key and 112-bit tweak.

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).

6.3 On the Round Function

The round function in our (tweakable) block cipher design is similar to the round function of `LED`. We have, however, introduced a small modification to the components of `LED` to assure maximal friendliness to lightweight implementations. With this respect, we have changed the S-Box and the matrix in the diffusion layer. For the substitution layer, we have selected the same S-Box as in the `Piccolo` block cipher, which offers a very compact implementation in hardware. This S-Box has been selected in all the 4-bit S-Boxes as it is optimal for both linear and differential cryptanalysis. We note that it as no fixed point, and has an overall algebraic degree of 3. For the diffusion layer, we have also chosen a compact implementation on 4×4 MDS involutory matrix, which can therefore be used as is for both encryption and decryption.

Chapter 7

Hardware Performances

Due to time constraints, we do not provide hardware implementations of Joltik. However, we explain in this chapter why we believe Joltik is a very lightweight candidate and the logic behind our ASIC implementation estimation.

Since Joltik-BC is also a 64-bit lightweight AES-like cipher, our starting point is the best ASIC lightweight implementation [18] of LED-128, that only requires 1265 GE (from which 77% comes from the memory to store the key and the internal state). The main differences of Joltik-BC compared to LED-128 is that the Sbox in Joltik-BC is a bit more compact than in LED, but on the other hand the diffusion matrix can not be computed serially as it is the case for LED. This later point should imply 3 nibbles (12 bits) of extra memory for the computation, which we estimate to $12 \times 6 = 72$ GE (counting 6 GE for a two-bit input flip-flop). The coefficients used in the Joltik-BC matrix are very lightweight (all coefficients in a row can be implemented with only 6 XORs [22] thanks to the careful choice of the coefficients and the finite field), so this should not have much impact compared to LED. Both LED-128 and Joltik-BC-128 have the same tweakey size, and both use a very light tweakey schedule. The permutation for the tweakey in Joltik-BC-128 can be implemented with bit wiring so this should have no impact on the area figures. The main difference would be that 128 bits are XORed from the tweakey state to the internal state each round, while in the case of LED it is only 64 bits. Therefore, we have to add an extra $64 \times (2.67) = 171$ GE by counting 2.67 GE per XOR (which might sometimes be optimized to 2.33 GE). We omit the multiplication by 2 in $GF(2^4)/\texttt{0x13}$ since it can be implemented with only shifts and a single XOR. Overall, we expect an overhead of 243 GE for Joltik-BC-128 compared to LED-128. This would lead us to about 1500 GE to implement Joltik-BC-128.

The reasoning for Joltik-BC-192 is exactly the same, except that we have an extra $64 \times (2.67) = 171$ GE to compute the extra tweakey XOR in the internal state. Moreover, an additional memory of 64 bits of tweakey material is required, which adds another $64 \times (4.67) = 298$ (counting 4.67 GE for a single-bit input flip-flop). Finally, we omit the multiplication by 4 in $GF(2^4)/\texttt{0x13}$ since it can be implemented with only shifts and two XORs. Overall, we expect Joltik-BC-192 to fit in about 2000 GE.

Concerning the authenticated encryption mode for Joltik$^{\neq}$, one can remark that it calls directly Joltik-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 64-bit checksum needs to be computed and stored. Therefore, one should count an additional $64 \times (4.67 + 2.67) = 470$ GE.
- the tweak value needs to be increased every Joltik-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 `Joltik`$^{\neq}$ (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 64-bit authentication value needs to be maintained until the end, similarly to the checksum, and this would add another $64 \times (4.67 + 2.67) = 470$ GE. However, this can be simply avoided if the associated data is computed after the message blocks (by directly XORing the output of the `Joltik-BC` calls to the checksum register). Therefore, we do not add extra cost for this part.

All in all, we estimate that `Joltik`$^{\neq}$-64-64 should be able to fit in about 2100 GE, and `Joltik`$^{\neq}$-80-112, `Joltik`$^{\neq}$-96-96 and `Joltik`$^{\neq}$-128-64 in 2600 GE. Concerning `Joltik`$^{=}$, the reasoning is exactly the same, except that the 64-bit checksum value is not required anymore. Therefore, a saving of 470 GE can be achieved when compared to `Joltik`$^{\neq}$. We estimate that `Joltik`$^{=}$-64-64 should be able to fit in about 1600 GE, and `Joltik`$^{=}$-80-112, `Joltik`$^{=}$-96-96 and `Joltik`$^{=}$-128-64 in 2100 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 `Joltik` 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.

We also recall that if both encryption and decryption capabilities are required, `Joltik-BC` is a good candidate, as the diffusion matrix is involutory. The tweak and key schedule are also pretty much the same in encryption and decryption modes. These two aspects will minimize the decryption overhead.

Finally, since for `Joltik`$^{\neq}$-64-64 and `Joltik`$^{=}$-64-64, the key is never changed and used as is, it can be hard-wired in some applications permitting this feature in order to save 470 GE.

Chapter 8

Intellectual Property

Joltik 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 $\texttt{Joltik}^{\neq}$ uses a mode that presents similarities with the generic ΘCB3 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 9

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.

Acknowledgments

The authors would like to thank Tetsu Iwata, Guo Jian, Gaëtan Leurent and Wang Lei for very fruitful discussions on authenticated encryption designs. Moreover, we are very grateful to Christof Beierle for pointing to us an error in our first general formulation of the bound on the number of active Sboxes coming from the subtweakeys schedule in the `STK` construction. Finally, the authors would like to thank Yannick Seurin for his essential contribution in the design of the `SCT` mode. This work is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

Bibliography

[1] Benadjila, R., Guo, J., Lomné, V., Peyrin, T.: Implementing Lightweight Block Ciphers on x86 Architectures. IACR Cryptology ePrint Archive **2013** (2013) 445

[2] Benadjila, R., Guo, J., Lomné, V., Peyrin, T.: Implementing Lightweight Block Ciphers on x86 Architectures. SAC **2013** (2013) To appear.

[3] Bilgin, B., Bogdanov, A., Knežević, 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

[4] 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

[5] 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

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

[7] Bogdanov, A., Mendel, F., Regazzoni, F., Rijmen, V., Tischhauser, E.: ALE: AES-Based Lightweight Authenticated Encryption. In Moriai, S., ed.: FSE 2013. Volume 8424 of LNCS., Springer (March 2014) 447–466

[8] Canteaut, A., ed.: FSE 2012. In Canteaut, A., ed.: FSE 2012. Volume 7549 of LNCS., Springer (March 2012)

[9] Daemen, J., Knudsen, L.R., Rijmen, V.: The Block Cipher Square. In Biham, E., ed.: FSE'97. Volume 1267 of LNCS., Springer (January 1997) 149–165

[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] Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Key Recovery Attacks on 3-round Even-Mansour, 8-step LED-128, and Full AES2. In Sako, K., Sarkar, P., eds.: ASIACRYPT 2013, Part I. Volume 8269 of LNCS., Springer (December 2013) 337–356

[13] Dinur, I., Jean, J.: Cryptanalysis of FIDES. In Cid, C., Rechberger, C., eds.: FSE 2014. Volume 8540 of LNCS., Springer (March 2015) 224–240

[14] 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

[15] Fleischmann, E., Forler, C., Lucks, S.: McOE: A Family of Almost Foolproof On-Line Authenticated Encryption Schemes. [8] 196–215

[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. [36] 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. [8] 110–126

[21] Jean, J., Nikolic, I., Peyrin, T.: Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. In Sarkar, P., Iwata, T., eds.: ASIACRYPT 2014, Part II. Volume 8874 of LNCS., Springer (December 2014) 274–288

[22] Khoongming Khoo and Frédérique Oggier and Thomas Peyrin and Siang Meng Sim: Lightweight MDS Involution Matrices (2014) *Article in preparation.*

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

[24] Khovratovich, D., Rechberger, C.: The LOCAL Attack: Cryptanalysis of the Authenticated Encryption Scheme ALE. In Lange, T., Lauter, K., Lisonek, P., eds.: SAC 2013. Volume 8282 of LNCS., Springer (August 2014) 174–184

[25] 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

[26] Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. In Yung, M., ed.: CRYPTO 2002. Volume 2442 of LNCS., Springer (August 2002) 31–46

[27] Liskov, M., Rivest, R.L., Wagner, D.: Tweakable Block Ciphers. Journal of Cryptology **24**(3) (July 2011) 588–613

[28] 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

[29] 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

[30] Matsuda, S., Moriai, S.: Lightweight Cryptography for the Cloud: Exploit the Power of Bitslice Implementation. In Prouff, E., Schaumont, P., eds.: CHES 2012. Volume 7428 of LNCS., Springer (September 2012) 408–425

[31] 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)

[32] 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

[33] Nikolić, I., Wang, L., Wu, S.: Cryptanalysis of round-reduced LED. FSE. To appear in Lecture Notes in Computer Science (2013)

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

[35] Peyrin, T., Seurin, Y.: SCT Mode of Operation. *to appear.*

[36] Preneel, B., Takagi, T., eds.: CHES 2011. In Preneel, B., Takagi, T., eds.: CHES 2011. Volume 6917 of LNCS., Springer (September / October 2011)

[37] Rogaway, P., Shrimpton, T.: A Provable-Security Treatment of the Key-Wrap Problem. In Vaudenay, S., ed.: EUROCRYPT 2006. Volume 4004 of LNCS., Springer (May / June 2006) 373–390

[38] Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An Ultra-Lightweight Blockcipher. [36] 342–357

[39] 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)

[40] Wu, H.: Related-Cipher Attacks. In Deng, R.H., Qing, S., Bao, F., Zhou, J., eds.: ICICS. Volume 2513 of Lecture Notes in Computer Science., Springer (2002) 447–455

Appendix A

Constants Defined in Joltik-BC

A.1 S-Box used in Joltik-BC

0xE	0x4	0xB	0x2	0x3	0x8	0x0	0x9	0x1	0xA	0x7	0xF	0x6	0xC	0x5	0xD

Table A.1: The Piccolo S-Box $\mathcal{S}$ used in Joltik-BC.

0x6	0x8	0x3	0x4	0x1	0xE	0xC	0xA	0x5	0x7	0x9	0x2	0xD	0xF	0x0	0xB

Table A.2: The Piccolo S-Box $\mathcal{S}^{-1}$ used in Joltik-BC.

A.2 Constants in the Key Schedule of Joltik-BC

We define in Table A.3 the values $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ used in the tweakey scheduling algorithm of Joltik-BC for each round number i to create RC_i. The constants are given in terms of bytes, and rc_0 represents the less significant bit.

i	Values of $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ used to build RC_i
0-11	01, 03, 07, 0F, 1F, 3E, 3D, 3B, 37, 2F, 1E, 3C
12-23	39, 33, 27, 0E, 1D, 3A, 35, 2B, 16, 2C, 18, 30
24-32	21, 02, 05, 0B, 17, 2E, 1C, 38, 31

Table A.3: Constants used in the internal block cipher of Joltik-BC.

Appendix B

Changelog

B.1 Changelog from v1.2 to v1.3

We detail here the differences between v1.2 and v1.3 of this document.

1. the most important modification is the replacement of the `COPA`-based mode by the new Synthetic Counter in Tweak (`SCT`) mode when nonce-misuse resistance is required.
2. for the the nonce-respecting mode $\mathcal{E}^{\neq}$ and $\mathcal{D}^{\neq}$, the nonce N is removed from the tweak input during the processing of the associated data, offering considerable speed-up when the associated data is fixed.
3. made the security claims more precise.
4. changed the parameters of $\texttt{Joltik}^{\neq}$-80-48 and $\texttt{Joltik}^{=}$-80-48 to create $\texttt{Joltik}^{\neq}$-80-112 and $\texttt{Joltik}^{=}$-80-112 instead.

B.2 Changelog from v1.1 to v1.2

We detail here the differences between v1.1 and v1.2 of this document.

1. Removed one block cipher call in the associated data in the case this input is empty.

B.3 Changelog from v1 to v1.1

1. Complete specifications of the nonce-misuse resistant mode.
2. Website link added.
3. Acknowledgments section added.
4. Typos and minor inconsistencies corrected.
5. Absence of hidden weaknesses statement added.