SKINNY-AEAD and SKINNY-Hash

v1.1

Christof Beierle[1,4], Jérémy Jean[2], Stefan Kölbl[3], Gregor Leander[4], Amir Moradi[4], Thomas Peyrin[5], Yu Sasaki[6], Pascal Sasdrich[4,7], and Siang Meng Sim[5]

[1] SnT, University of Luxembourg, Luxembourg
beierle.christof@gmail.com

[2] ANSSI, Paris, France
Jean.Jeremy@gmail.com

[3] Cybercrypt A/S, Denmark
kste@mailbox.org

[4] Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Germany
{Firstname.Lastname}@rub.de

[5] School of Physical and Mathematical Sciences
Nanyang Technological University, Singapore
Thomas.Peyrin@ntu.edu.sg, crypto.s.m.sim@gmail.com

[6] NTT Secure Platform Laboratories, Japan
Sasaki.Yu@lab.ntt.co.jp

[7] Rambus Cryptography, The Netherlands

Table of Contents

1 Introduction

In this document, we specify our submissions SKINNY-AEAD and SKINNY-Hash to the NIST lightweight cryptography standardization process. SKINNY is a family of lightweight tweakable block ciphers proposed at CRYPTO 2016 [7]. We specify how to provide the AEAD and hashing functionalities by using SKINNY as a base primitive.

SKINNY-AEAD. In short, SKINNY-AEAD uses a mode following the general ΘCB3 framework, instanciated with SKINNY-128-384. The fact that SKINNY is a beyond birthday-bound secure tweakable block cipher enables to achieve the provable security providing *full* security in the nonce-respecting setting. A similar mode was also employed in the third-round CAESAR candidate Deoxys-I [22]. Our primary design takes a 128-bit key, a 128-bit nonce, and an associated data and a message of up to $2^{64} \times 16$ bytes. It then outputs a ciphertext of the same length as the plaintext and a 128-bit tag. We also specify other members of this family to support any combination of n_ℓ- and t_ℓ-bit nonces and tags, respectively, where $n_\ell \in \{96, 128\}$ and $t_\ell \in \{64, 128\}$.

Moreover, we also specify the lightweight version instantiated with SKINNY-128-256. This design is motivated from the observation that the submission requirement to support 2^{50} input bytes might be unnecessary for several of the use cases of the lightweight cryptography. This family consists of two members that take a 128-bit key, a 96-bit nonce, and an associated data and a message of up to 2^{28} bytes as input and produce the ciphertext and a t_ℓ-bit tag, where $t_\ell \in \{64, 128\}$. Because of the restriction of the maximum number of input message bytes, this family do not satisfy the submission requirement to support input messages of up to 2^{50} bytes, yet provides even smaller and faster AEAD schemes.

SKINNY-Hash. SKINNY-Hash consists of two members of the hash function schemes that adopt the well-known sponge construction.

Our primary member uses a 384-bit to 384-bit *function* built with SKINNY-128-384 to provide a 128-bit secure hash function and the secondary member uses a 256-bit to 256-bit *function* built with SKINNY-128-256 to provide a 112-bit secure hash function.

Features. Before going into details of the specifications, we briefly summarize the main features of our design in the following.

- **Well-understood design and high security margin.** The SKINNY family of tweakable block ciphers was designed as a solid Substitution-Permutation network (SPN) having a well-analyzed security bound against the most fundamental cryptanalytic approaches: differential cryptanalysis and linear cryptanalysis. In addition, SKINNY receives a lot of security analysis by third-party researchers, which demonstrates its strong resistance against cryptanalysis. The cipher can basically be understood as a tweakable version of a *tailored* AES which omits or simplifies all components not strictly necessary for the security. Therefore, similar cryptanalytic approaches as for AES can be applied to our design. However, opposed to AES, the TWEAKEY framework allows to derive strong security arguments in the related-key, related-tweak setting for SKINNY. Moreover, SKINNY offers a high security margin. At the time of submission, based on our own cryptanalysis and the extensive external cryptanalysis since its publication, SKINNY-128-384 offers 29 (out of 56) rounds, and SKINNY-128-256 offers 25 (out of 48) rounds of security margin in the related-tweakey setting. In other words, less than half of the number of rounds have been broken so far.
- **Security proofs by a modular approach.** The security of the authenticated encryption schemes and hash functions are directly inherited from the well-known and widely-applied modes of operation used in our design. Indeed, SKINNY-AEAD relies on

the proofs of the ΘCB3 mode, while for SKINNY-Hash we rely on the provable security of the sponge framework. The security of our schemes can thus be reduced to the ideal behavior of the underlying primitives SKINNY-128-384 and SKINNY-128-256.

- **Beyond-birthday-bound security.** By using a tweakable block cipher directly constructed by the TWEAKEY framework, we obtain beyond-birthday-bound security which allows to efficiently exploit the whole state. This is different to modes based on OCB, which only offers security up to the birthday bound. Such modes would require larger internal states to achieve the same security level.
- **Efficient protection against side-channel attacks.** Thanks to the structured Sbox of SKINNY, which is an iteration of a quadratic permutation, its Threshold Implementation (a provably-secure countermeasure against side-channel analysis attacks) can be efficiently made. This helps us to efficiently integrate side-channel countermeasures into various implementations of SKINNY with minimum number of shares and limited number of fresh randomness, both affecting the area overhead of the resulting design.
- **General-purpose lightweight schemes.** When designing a lightweight encryption or hashing scheme, several use cases must be taken in account. While area-optimized implementations are important for some very constrained applications, throughput or throughput-over-area optimized implementations are also very relevant. Actually, looking at recently introduced efficiency measurements [24], one can see that our designs choices are good for many types of implementations, which is exactly what makes a good general-purpose lightweight encryption scheme.
- **Efficiency for short messages.** Our algorithms are efficient for short messages. For authenticated encryption, the main reason is because the design is based on a tweakable block cipher, which allows to avoid any precomputation (like in OCB, AES-GCM, etc.). In particular, the first 128-bit message block is handled directly and by taking in account the tag generation, one needs only $m + 1$ internal calls to the tweakable block cipher to process messages of m blocks of 128 bits each (if there is no associated data).
  Our primary member for hashing requires at most $3(m + 2)$ calls to the tweakable block cipher for producing a 256-bit digest for a message of m blocks of 128 bits each.
- **Parallelizable mode.** Our AEAD schemes are fully parallilizable as they are based on the ΘCB3 mode, which employs independent calls to the tweakable block cipher.
- **Flexibility.** Our AEAD design allows for smooth parameter handling. For the official NIST submision given in this document we define specific parameter sets, but any user can in principle choose its own separation into nonce, key and block counter 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. In a nutshell, one implementation of the underlying cipher is sufficient to support all versions with different key and tweak sizes (which sum up to the same size).

2 Specification

2.1 Notations

By $\|$ we denote the concatenation of bitstrings. Given a bitstring B, we denote by B^j the j-times concatenation of B, where B^0 is defined to be the empty string ϵ. For instance $0\|10^3 = 010^3 = 01000$ and $(10)^3 = 101010$. We denote the length of a string B in bits by $|B|$, where $|\epsilon| = 0$.

2.2 Parameter Sets

AEAD. In a nutshell, our AEAD scheme adopts a mode that can be described in the ΘCB3 framework [27] by using either SKINNY-128-384 or SKINNY-128-256 as the underlying tweakable block cipher.

Our primary member instantiates ΘCB3 with the tweakable block cipher SKINNY-128-384 used with 128-bit keys, 128-bit nonces and which produces 128-bit authentication tags. Along with this primary AEAD scheme, we propose three additional ones that extend the possible parameters and allow users to choose between two nonce sizes (96 bits or 128 bits), and two tag sizes (128 bits or 64 bits). All of them are consistent with NIST's requirements.

We also specify two secondary options that are designed for processing short inputs. Those are based on a second tweakable block cipher, namely SKINNY-128-256. The nonce size is fixed to 96 bits, while users can choose between two tag sizes: 128 or 64 bits. The maximum number of message blocks that can be processed with SKINNY-128-256-based members is limited to 2^{28} bytes. Users need to be careful about its usage because these two algorithms *do not meet* NIST's requirements to support input messages of up to 2^{50} bytes.

Hashing. Overall, the SKINNY-Hash family contains two function-based sponge constructions [11], in which the underlying functions are built with SKINNY-128-384 and SKINNY-128-256. Both family members, denoted SKINNY-tk3-Hash and SKINNY-tk2-Hash, process input messages of arbitrary length and output a 256-bit digest.

A list of our proposed AEAD schemes (members M1 to M6), together with the two hashing algorithms is provided in Table 1. For comparisons, we pair the AEAD members M1, M2, M3 and M4 with the hashing algorithm SKINNY-tk3-Hash and the AEAD members M5 and M6 with SKINNY-tk2-Hash as the constructions in the respective pairs are based on the same variant of the SKINNY tweakable block cipher.

Table 1: Our proposed AEAD schemes and hashing algorithms.

Member	Block Cipher	Nonce	Tag	Key	Hash Function	Rate	Capacity
	underlying primitive	bits	bits	bits	type	bits	bits
M1 †	SKINNY-128-384	128	128	128	384-bit sponge	128	256
M2	SKINNY-128-384	96	128	128			
M3	SKINNY-128-384	128	64	128			
M4	SKINNY-128-384	96	64	128			
M5 *	SKINNY-128-256	96	128	128	256-bit sponge	32	224
M6 *	SKINNY-128-256	96	64	128			

†: Primary member. *: Do not strictly follow NIST requirements.

2.3 The Tweakable Block Ciphers SKINNY-128-256 and SKINNY-128-384

We already published the SKINNY family of tweakable block ciphers in 2016 in [7]. For the sake of completeness, we provide the specifications of the two members of the SKINNY family that are relevant for this submission, namely SKINNY-128-256 and SKINNY-128-384 below.

The tweakable block ciphers SKINNY-128-256 and SKINNY-128-384 both have a block size of $n = 128$ bit and the internal state is viewed as a 4×4 square array of cells, where each cell contains a byte. We denote $IS_{i,j}$ the cell of the internal state located at Row i and Column j (counting starts from 0). One can also view this 4×4 square array of cells as a vector of cells by concatenating the rows. Thus, we denote with a single subscript IS_i the cell of the internal state located at Position i in this vector (counting starts from 0) and we have that $IS_{i,j} = IS_{4 \cdot i+j}$.

The ciphers follow the TWEAKEY framework from [21] and therefore take a tweakey input – instead of a key only – without any distinction between key and tweak input.

The two tweakable block ciphers SKINNY-128-256 and SKINNY-128-384 mainly differ in the size of the tweakey input: they respectively process $2n = 256$ or $3n = 384$ tweakey bits. The tweakey state is also viewed as a collection of two (resp., three) 4×4 square arrays of cells of 8 bits each. We denote these arrays $TK1$ and $TK2$ for SKINNY-128-256 and $TK1$, $TK2$ and $TK3$ for SKINNY-128-384. Moreover, we denote $TKz_{i,j}$ the cell of the tweakey state located at Row i and Column j of the z-th cell array. As for the internal state, we extend this notation to a vector view with a single subscript: $TK1_i$, $TK2_i$ and $TK3_i$.

We now give the structural specifications of the ciphers.

Initialization. The ciphers receive a plaintext $m = m_0 \| m_1 \| \cdots \| m_{14} \| m_{15}$, where the m_i are 8-bit words. The initialization of the ciphers' internal state are performed by simply setting $IS_i = m_i$ for $0 \leq i \leq 15$, i.e.,

$$IS = \begin{bmatrix} m_0 & m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 & m_7 \\ m_8 & m_9 & m_{10} & m_{11} \\ m_{12} & m_{13} & m_{14} & m_{15} \end{bmatrix}.$$

Note that the state is loaded row-wise rather than in the column-wise fashion as done for example in the AES. This is a more hardware-friendly choice, as pointed out in [33].

The ciphers receive a tweakey input $tk = tk_0 \| tk_1 \| \cdots \| tk_{31}$ (resp., $tk = tk_0 \| tk_1 \| \cdots \| tk_{47}$), where the tk_i are 8-bit words. The initialization of the cipher's tweakey state is performed by simply setting for $0 \leq i \leq 15$:

For $2n$-bit tweakey:	For $3n$-bit tweakey:
$TK1_i = tk_i$	$TK1_i = tk_i$
$TK2_i = tk_{16+i}$	$TK2_i = tk_{16+i}$
	$TK3_i = tk_{32+i}$

Note that the tweakey states are also loaded row-wise.

Round Function. One encryption round of SKINNY is composed of five operations in the following order: SubCells, AddConstants, AddRoundTweakey, ShiftRows and MixColumns (see illustration in Figure 1). The number r of rounds to perform during encryption depends on the tweakey size. In particular, SKINNY-128-256 applies $r = 48$ and SKINNY-128-384 applies $r = 56$ rounds. Note that no whitening key is used.

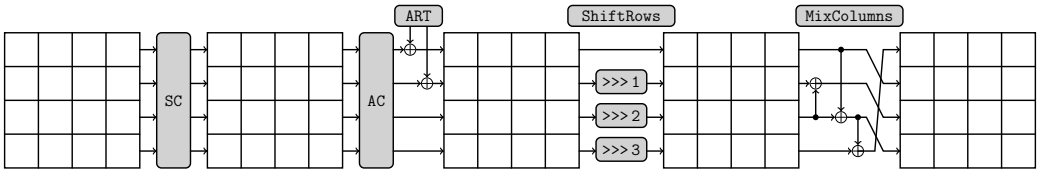


Figure 1: The SKINNY round function applies five different transformations: SubCells (SC), AddConstants (AC), AddRoundTweakey (ART), ShiftRows (SR) and MixColumns (MC).

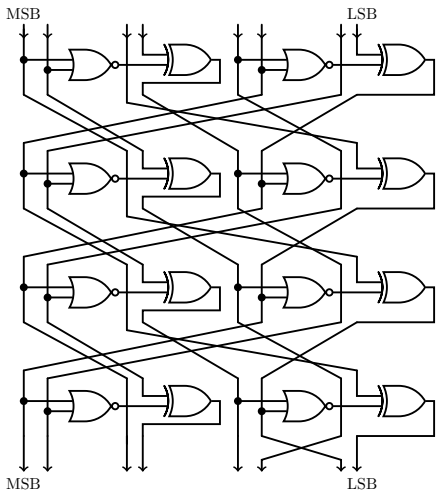


Figure 2: Construction of the Sbox $\mathcal{S}_8$.

SubCells. An 8-bit Sbox $\mathcal{S}_8$ is applied to every cell of the ciphers internal state. Its design, depicted in Figure 2, is simple and inspired by the PICCOLO Sbox [43].
If $x_0, \ldots, x_7$ represent the eight inputs bits of the Sbox (x_0 being the least significant bit), it basically applies the below transformation on the 8-bit state:

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \rightarrow (x_7, x_6, x_5, x_4 \oplus (\overline{x_7 \vee x_6}), x_3, x_2, x_1, x_0 \oplus (\overline{x_3 \vee x_2})),$$

followed by the bit permutation

$$(x_7, x_6, x_5, x_4, x_3, x_2, x_1, x_0) \longrightarrow (x_2, x_1, x_7, x_6, x_4, x_0, x_3, x_5),$$

repeating this process four times, except for the last iteration where there is just a bit swap between x_1 and x_2. Besides, we provide in Appendix A the table of $\mathcal{S}_8$ and its inverse in hexadecimal notations.

AddConstants. A 6-bit affine LFSR, whose state is denoted (rc_5, rc_4, rc_3, rc_2, rc_1, rc_0) (with rc_0 being the least significant bit), is used to generate round constants. Its update function is defined as:

$$(\mathrm{rc}_5||\mathrm{rc}_4||\mathrm{rc}_3||\mathrm{rc}_2||\mathrm{rc}_1||\mathrm{rc}_0) \rightarrow (\mathrm{rc}_4||\mathrm{rc}_3||\mathrm{rc}_2||\mathrm{rc}_1||\mathrm{rc}_0||\mathrm{rc}_5 \oplus \mathrm{rc}_4 \oplus 1)\ .$$

The six bits are initialized to zero, and updated *before* used in a given round. The bits from the LFSR are arranged into a 4×4 array and only the first column of the state

is affected by the LFSR bits, i.e.,

$$\begin{bmatrix} c_0 & 0 & 0 & 0 \\ c_1 & 0 & 0 & 0 \\ c_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

with $c_2 = \texttt{0x2}$ and $(c_0, c_1) = (0\|0\|0\|0\|\mathrm{rc}_3\|\mathrm{rc}_2\|\mathrm{rc}_1\|\mathrm{rc}_0,\ 0\|0\|0\|0\|0\|0\|\mathrm{rc}_5\|\mathrm{rc}_4)$. The round constants are combined with the state, respecting array positioning, using bitwise exclusive-or. The values of the $(\mathrm{rc}_5, \mathrm{rc}_4, \mathrm{rc}_3, \mathrm{rc}_2, \mathrm{rc}_1, \mathrm{rc}_0)$ constants for each round are given in Table 2 below, encoded to byte values for each round, with rc_0 being the least significant bit.

Table 2: `SKINNY` Round Constants.

Rounds	Constants
1 - 16	`01,03,07,0F,1F,3E,3D,3B,37,2F,1E,3C,39,33,27,0E`
17 - 32	`1D,3A,35,2B,16,2C,18,30,21,02,05,0B,17,2E,1C,38`
33 - 48	`31,23,06,0D,1B,36,2D,1A,34,29,12,24,08,11,22,04`
49 - 62	`09,13,26,0C,19,32,25,0A,15,2A,14,28,10,20`

`AddRoundTweakey`. The first and second rows of all tweakey arrays are extracted and bitwise exclusive-ored to the ciphers internal state, respecting the array positioning. More formally, for $i = \{0, 1\}$ and $j = \{0, 1, 2, 3\}$, we have:

- $IS_{i,j} = IS_{i,j} \oplus TK1_{i,j} \oplus TK2_{i,j}$ for `SKINNY-128-256`,
- $IS_{i,j} = IS_{i,j} \oplus TK1_{i,j} \oplus TK2_{i,j} \oplus TK3_{i,j}$ for `SKINNY-128-384`.

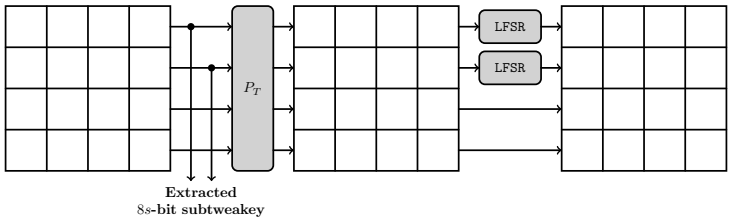


Figure 3: The tweakey schedule in `SKINNY`. Each tweakey word $TK1$, $TK2$ and $TK3$ (if any) follows a similar transformation update, except that no LFSR is applied to $TK1$.

Then, the tweakey arrays are updated as follows (this tweakey schedule is illustrated in Figure 3). First, a permutation P_T is applied on the cells positions of all tweakey arrays: for all $0 \leq i \leq 15$, we set $TKz_i \leftarrow TKz_{P_T[i]}$ with

$$P_T = [9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7],$$

for $z \in \{1, 2\}$ (resp., $z \in \{1, 2, 3\}$). This corresponds to the following reordering of the matrix cells, where indices are taken row-wise:

$$(0, \ldots, 15) \xmapsto{P_T} (9, 15, 8, 13, 10, 14, 12, 11, 0, 1, 2, 3, 4, 5, 6, 7).$$

Table 3: The LFSRs used in SKINNY to generate the round tweakeys. The TK parameter gives the number of the corresponding tweakey word in the cipher.

TK	s	LFSR
$TK2$	8	$(x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0) \rightarrow (x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0\|\|x_7 \oplus x_5)$
$TK3$	8	$(x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1\|\|x_0) \rightarrow (x_0 \oplus x_6\|\|x_7\|\|x_6\|\|x_5\|\|x_4\|\|x_3\|\|x_2\|\|x_1)$

Finally, every cell of the first and second rows of $TK2$ (resp., $TK2$ and $TK3$) are individually updated with an LFSR. The LFSRs used are given in Table 3 (x_0 stands for the LSB of the cell).

`ShiftRows`. As in the `AES`, in this layer, the rows of the cipher state cell array are rotated. More precisely, the second, third, and fourth cell rows are rotated by 1, 2 and 3 positions *to the right*, respectively. In other words, a permutation P is applied on the cells positions of the cipher internal state cell array: for all $0 \leq i \leq 15$, we set $IS_i \leftarrow IS_{P[i]}$ with

$$P = [0, 1, 2, 3, 7, 4, 5, 6, 10, 11, 8, 9, 13, 14, 15, 12].$$

`MixColumns`. Each column of the cipher internal state array is multiplied by the following binary matrix $\mathbf{M}$:

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}.$$

The final value of the internal state array provides the ciphertext with cells being unpacked in the same way as the packing during initialization. Test vectors for `SKINNY-128-256` and `SKINNY-128-384` are provided in Appendix B. Note that decryption is very similar to encryption as all cipher components have very simple inverses (`SubCells` and `MixColumns` are based on a generalized Feistel structure, so their respective inverse is straightforward to deduce and can be implemented with the exact same number of operations).

2.4 The AEAD Scheme SKINNY-AEAD

The authenticated encryption scheme adopts the ΘCB3 mode using either `SKINNY-128-384` or `SKINNY-128-256` as the underlying tweakable block cipher, depending on the member as shown in Table 1. In the following, we provide the detailed specification of the scheme. Let $\texttt{SKINNY-128-384}_{tk}(P)$ denote the encryption of a plaintext P under the tweakey tk with the `SKINNY-128-384` algorithm and let $\texttt{SKINNY-128-256}_{tk}(P)$ be the encryption of a plaintext P under the tweakey tk with the `SKINNY-128-256` algorithm. Let further $\texttt{SKINNY-128-384}^{-1}_{tk}(C)$ (resp. $\texttt{SKINNY-128-256}^{-1}_{tk}(C)$) denote the decryption of a ciphertext C under the tweakey tk with the `SKINNY-128-384` (resp. `SKINNY-128-256`) algorithm.

By (N, A, M), we denote the tuple of a nonce N, associated data A and a message M, where A and M can be of arbitrary length (including empty).

`SKINNY-AEAD` Based on `SKINNY-128-384`. This case applies to our primary member (M1), as well as M2, M3, and M4 (refer to Table 1).

Domain Separation. We first define a 1-byte string that ensures independence of tweakable block cipher calls for different kinds of computations (i.e., *domain separation*) and also for different `SKINNY-AEAD` members. Let $b_7\|b_6\|b_5\|b_4\|b_3\|b_2\|b_1\|b_0$ be the bitwise representation of this byte, where b_7 is the MSB and b_0 is the LSB (see also Figure 4). Then, we use the following convention:

- b_7 to b_5 are always fixed to 0,
- b_4 encodes the nonce size $n_\ell \in \{0, 1\}$, where n_ℓ is set to 0 if the nonce size is 128 bits and 1 if the nonce size is 96 bits,
- b_3 encodes the tag size $t_\ell \in \{0, 1\}$, where t_ℓ is set to 0 if the tag size is 128 bits and 1 if the tag size is 64 bits,
- b_2 to b_0 are used for the actual domain separation, which is further specified as follows:
  - 000: encryption of a full message block,
  - 001: encryption of a partial message block,
  - 010: computation of a full associated data block,
  - 011: computation of a partial associated data block,
  - 100: generation of a tag if the message size in bits is a multiple of 128,
  - 101: generation of a tag if the message size in bits is not a multiple of 128.

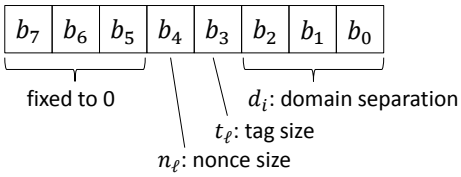


Figure 4: Domain separation and distinction of the different members.

Note that the nonce size (b_4) and the tag size (b_3) are fixed during the computation of a single tuple of nonce N, associated data A and message M, while b_2 to b_0 vary accross a computation of a single (N, A, M).

Hereafter, we specify the computations of a ciphertext C and a tag `tag` for a given (N, A, M), key K, n_ℓ, and t_ℓ. For simplicity, we denote this single byte by d_0, d_1, d_2, d_3, d_4, or d_5 depending on the 3-bit value for the domain separation, i.e.:

$$\begin{aligned} d_0 &= 000 n_\ell t_\ell 000, \\ d_1 &= 000 n_\ell t_\ell 001, \\ d_2 &= 000 n_\ell t_\ell 010, \\ d_3 &= 000 n_\ell t_\ell 011, \\ d_4 &= 000 n_\ell t_\ell 100, \\ d_5 &= 000 n_\ell t_\ell 101. \end{aligned}$$

Associated Data Processing. The computation for the associated data is depicted in Figure 5. If the byte-length of A is not a multiple of the block size (i.e., 16 bytes), it has to be padded. In particular, if $|A|$ denotes the length of A in bit, let $A = A_0\|A_1\|\ldots\|A_{\ell_a-1}\|A_{\ell_a}$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$. Note that if $|A|$ is a multiple of 128, $|A_{\ell_a}|$ is set to the empty string ϵ and no padding is applied. Otherwise, we apply the padding pad10* to A_{ℓ_a} which is defined as

$$\text{pad10*}\colon X \mapsto X\|1\|0^{127-|X| \bmod 128}.$$

Each associated data block A_i is processed in parallel by `SKINNY-128-384` as a plaintext input under a 384-bit tweakey, where the structure of the 384-bit tweakey is as follows.

- The tweakey bytes $tk_0, \ldots, tk_{15}$ store 8 bytes from a 64-bit LFSR, followed by 7 bytes of zeros, and then the single byte for the domain separation (d_2 or d_3 whether it is a padded block). The 64-bit LFSR plays the role of a block counter. It is defined as follows: Let $x_{63}\|x_{62}\|x_{61}\|\cdots\|x_2\|x_1\|x_0$ denote the 64-bit state of the LFSR. It is initialized to $\mathrm{LFSR}_0 = 0^{63}\|1$ and updated as $\mathrm{LFSR}_{t+1} = \mathrm{upd}_{64}(\mathrm{LFSR}_t)$, where the update function upd_{64} is defined by the polynomial $x^{64} + x^4 + x^3 + x + 1$ as

$$\mathrm{upd}_{64}\colon x_{63}\|x_{62}\|\ldots\|x_1\|x_0 \quad \longrightarrow \quad y_{63}\|y_{62}\|\ldots\|y_1\|y_0$$

  with:

$$\begin{aligned} &y_i \leftarrow x_{i-1} \text{ for } i \in \{63, 62, \ldots, 1\}\backslash\{4, 3, 1\},\\ &y_4 \leftarrow x_3 \oplus x_{63},\\ &y_3 \leftarrow x_2 \oplus x_{63},\\ &y_1 \leftarrow x_0 \oplus x_{63},\\ &y_0 \leftarrow x_{63}. \end{aligned}$$

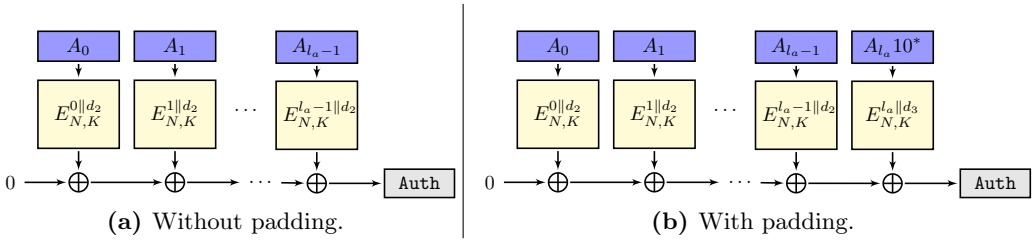
  Before loaded in the tweakey state, the order of the bytes of the LFSR state is reversed, i.e., $tk_0\|tk_1\|\ldots\|tk_{15} = \mathrm{rev}_{64}(\mathrm{LFSR}_t)\|0^{56}\|d_2$ (resp. $tk_{15} = d_3$ for the last padded block), where rev_{64} is defined by

$$\mathrm{rev}_{64}\colon x_7\|x_6\|x_5\|x_4\|x_3\|x_2\|x_1\|x_0 \mapsto x_0\|x_1\|x_2\|x_3\|x_4\|x_5\|x_6\|x_7 \quad (\forall i\colon |x_i| = 8)\ .$$

- The tweakey bytes $tk_{16}\|tk_{17}\|\ldots\|tk_{31}$ store the nonce N. If $n_\ell = 1$, i.e., if the nonce size is 96 bits, 32-bit zeros are appended to N, thus, $tk_{16}\|tk_{17}\|\ldots\|tk_{31} = N\|0^{32}$.
- The tweakey bytes $tk_{32}\|tk_{33}\|\ldots\|tk_{47}$ store the 128-bit key K.

The XOR sum of the each block's output is stored as `Auth`, which is later used in the final authentication tag computation.

Remind that if the size of A is not a multiple of 128 bits, we use the domain separation byte d_3 to process the last padded block.



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

Figure 5: Handling of the associated data: in the case where the associated data is a multiple of the block size, no padding is needed. In the figures, E refers to `SKINNY-128-384`. For simplicity, we denote the block counter by $0\ldots, \ell_a - 1$ (resp., $0, \ldots, \ell_a$) but actually refer to the state of the LFSRs serving as a block counter.

Encryption. The encryption of M is depicted in Figure 6 and Figure 7. First, suppose that the size of M in bits is a multiple of 128 (Figure 6). In that case, M is parsed into 128-bit blocks $M_0, M_1, M_2, \ldots, M_{\ell_m-1}$ and no padding is applied. Each message block M_i is processed by `SKINNY-128-384` as a plaintext input under a particular 384-bit tweakey and the output is taken as the corresponding ciphertext block. The structure of the 384-bit

tweakey differs from the associated data processing only by the domain separation byte. Here, the byte tk_{15} is fixed to d_0 instead of d_2.

To produce the tag, the XOR sum of the plaintext blocks noted Σ is computed and then encrypted by `SKINNY-128-384`, where the 384-bit tweakey is analogously defined as $tk_0\|tk_1\|\ldots\|tk_{47} = \mathrm{rev}_{64}(\mathrm{LFSR}_{\ell_m})\|0^{56}\|d_4\|N\|K$. Finally, the output of this encryption is XORed with `Auth`. If $t_\ell = 0$, i.e., the tag size is 128 bits, the result of this XOR is a tag. If $t_\ell = 1$, i.e., the tag size is 64 bits, the result of this XOR is truncated by trunc_{64} to 64 bit, where the truncation functions trunc_i are defined for inputs of length at least i by

$$\mathrm{trunc}_i\colon X = x_0\|x_1\|\ldots\|x_{|X|-1} \mapsto x_0\|x_1\|\ldots\|x_{i-1}.$$

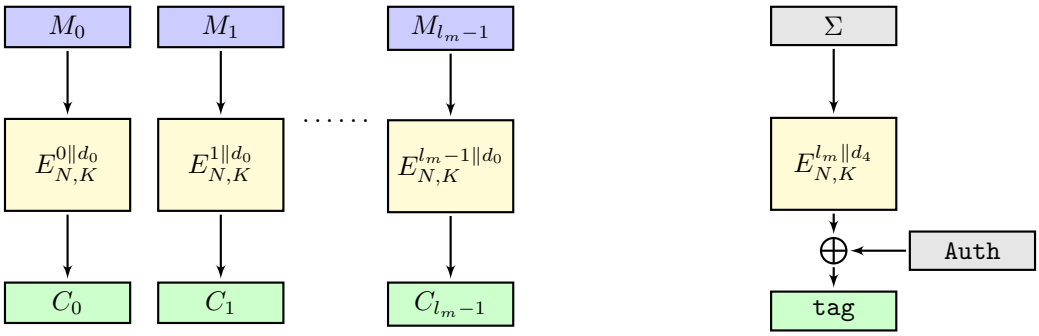


Figure 6: Encryption of `SKINNY-AEAD` with `SKINNY-128-384` without padding when $t_\ell = 128$. Again, E refers to `SKINNY-128-384`. For simplicity, we denote the block counter by $0,\ldots,\ell_m$ but actually refer to the state of the LFSRs serving as a block counter.

In the case $|M|$ is not a multiple of 128 (Figure 7), the same padding pad10* as for the associated data is applied to M. In particular, M is split into $M = M_0\|M_1\|\ldots\|M_{\ell_m-1}\|M_{\ell_m}$ with $|M_i| = 128$ for $i \in \{0,\ldots,\ell_m-1\}$ and $0 < |M_{\ell_m}| < 127$. The processing of the message blocks $M_i, i \in \{0,\ldots,\ell_m-1\}$ is the same as in the case described above. The last ciphertext block C_{ℓ_m} is computed as the XOR sum of the encryption of 0 with `SKINNY-128-384` under the 384-bit tweakey $tk_0\|tk_1\|\ldots\|tk_{47} = \mathrm{rev}_{64}(\mathrm{LFSR}_{\ell_m})\|0^{56}\|d_1\|N\|K$ (truncated to $|M_{\ell_m}|$ bits) with the plaintext block M_{ℓ_m}.

For the tag computation, the checksum is computed as $M_0 \oplus M_1 \oplus \cdots \oplus M_{\ell_m-1} \oplus \mathrm{pad10}^*(M_{\ell_m})$ and it is encrypted with `SKINNY-128-384` under the 384-bit tweakey

$$\mathrm{rev}_{64}(\mathrm{LFSR}_{\ell_{m+1}})\|0^{56}\|d_5\|N\|K.$$

Similar as for the unpadded case, the encryption is XORed with `Auth` and truncated in the same way as described above if $t_\ell = 1$.

Decryption. The decryption and tag verification procedure for given $(K, N, A, C, \mathsf{tag})$ is straightforward.

Formally, we provide the algorithms of the authenticated encryption members M1, M2, M3, and M4, together with their decryption and tag verification procedure, in Algorithms 1, 2, 3, 4, 5, 6 and 7, 8, respectively.

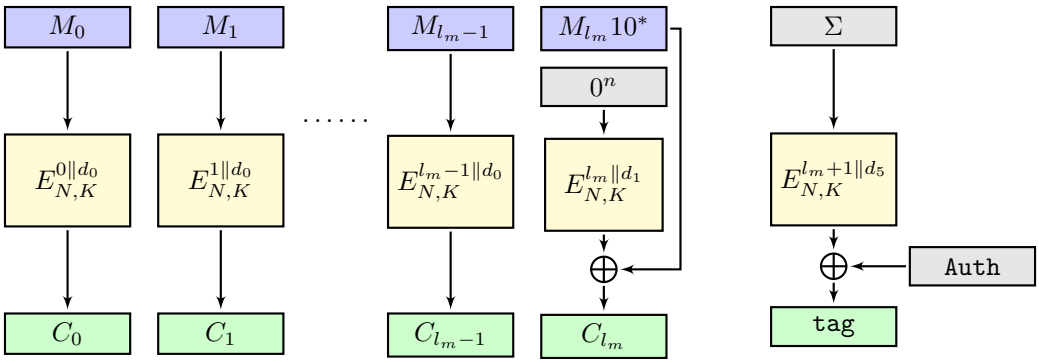


Figure 7: Encryption of SKINNY-AEAD with SKINNY-128-384 with padded message when $t_\ell = 128$. The last cipherblock block C_{l_m} is further truncated to have the same size as M_{l_m}. Again, we denote the block counter by $0, \ldots, \ell_m + 1$ but actually refer to the state of the LFSRs serving as a block counter.

Algorithm 1 The authenticated encryption algorithm $\texttt{SKINNY-AEAD-M1-Enc}(K, N, A, M)$
In: Key K, nonce N (both 128 bit), associated data A, message M (both arbitrarily long)
Out: (C, tag), where C is the ciphertext with $|C| = |M|$ and tag is a 128-bit tag

---

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000010 \| N \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000011 \| N \| K}(\mathrm{pad10^*}(A_{\ell_a}))$
//Encryption
$M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m} \leftarrow M$ with $|M_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|M_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $C_i \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000000 \| N \| K}(M_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $M_{\ell_m} = \epsilon$ **then**
  $C_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000100 \| N \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000001 \| N \| K}(0^{128})$
  $C_{\ell_m} \leftarrow M_{\ell_m} \oplus \mathrm{trunc}_{|M_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10^*}(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000101 \| N \| K}(\Sigma)$
$C \leftarrow C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m}$
//Tag generation
$\mathsf{tag} \leftarrow T \oplus \mathsf{Auth}$
return (C, tag)

Algorithm 2 The decryption algorithm $\texttt{SKINNY-AEAD-M1-Dec}(K, N, A, C, \mathsf{tag})$
In: Key K, nonce N (both 128 bit), associated data A, ciphertext C (both arbitrarily long), 128-bit tag tag
Out: M if tag is valid, $\perp$ otherwise

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000010 \| N \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000011 \| N \| K}(\mathrm{pad10^*}(A_{\ell_a}))$
//Decryption
$C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m} \leftarrow C$ with $|C_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|C_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $M_i \leftarrow \texttt{SKINNY-128-384}^{-1}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000000 \| N \| K}(C_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $C_{\ell_m} = \epsilon$ **then**
  $M_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000100 \| N \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000001 \| N \| K}(0^{128})$
  $M_{\ell_m} \leftarrow C_{\ell_m} \oplus \mathrm{trunc}_{|C_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10^*}(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00000101 \| N \| K}(\Sigma)$
$M \leftarrow M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m}$
//Tag verification
$\mathsf{tag}' \leftarrow T \oplus \mathsf{Auth}$
if $\mathsf{tag}' = \mathsf{tag}$ **then**
  return M
else
  return $\perp$

Algorithm 3 The authenticated encryption algorithm $\texttt{SKINNY-AEAD-M2-Enc}(K, N, A, M)$
In: 128-bit key K, 96-bit nonce N, associated data A, message M (both arbitrarily long)
Out: (C, tag), where C is the ciphertext with $|C| = |M|$ and tag is a 128-bit tag

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010010 \| N \| 0^{32} \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010011 \| N \| 0^{32} \| K}(\mathrm{pad10}^*(A_{\ell_a}))$
//Encryption
$M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m} \leftarrow M$ with $|M_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|M_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $C_i \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010000 \| N \| 0^{32} \| K}(M_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $M_{\ell_m} = \epsilon$ **then**
  $C_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010100 \| N \| 0^{32} \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010001 \| N \| 0^{32} \| K}(0^{128})$
  $C_{\ell_m} \leftarrow M_{\ell_m} \oplus \mathrm{trunc}_{|M_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10}^*(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010101 \| N \| 0^{32} \| K}(\Sigma)$
$C \leftarrow C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m}$
//Tag generation
$\mathsf{tag} \leftarrow T \oplus \mathsf{Auth}$
return (C, tag)

Algorithm 4 The decryption algorithm $\texttt{SKINNY-AEAD-M2-Dec}(K, N, A, C, \mathsf{tag})$
In: 128-bit key K, 96-bit nonce N, associated data A, ciphertext C (both arbitrarily long), 128-bit tag tag
Out: M if tag is valid, $\perp$ otherwise

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010010 \| N \| 0^{32} \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010011 \| N \| 0^{32} \| K}(\mathrm{pad10}^*(A_{\ell_a}))$
//Decryption
$C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m} \leftarrow C$ with $|C_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|C_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $M_i \leftarrow \texttt{SKINNY-128-384}^{-1}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010000 \| N \| 0^{32} \| K}(C_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $C_{\ell_m} = \epsilon$ **then**
  $M_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010100 \| N \| 0^{32} \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010001 \| N \| 0^{32} \| K}(0^{128})$
  $M_{\ell_m} \leftarrow C_{\ell_m} \oplus \mathrm{trunc}_{|C_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10}^*(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00010101 \| N \| 0^{32} \| K}(\Sigma)$
$M \leftarrow M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m}$
//Tag verification
$\mathsf{tag}' \leftarrow T \oplus \mathsf{Auth}$
if $\mathsf{tag}' = \mathsf{tag}$ **then**
  return M
else
  return $\perp$

Algorithm 5 The authenticated encryption algorithm $\texttt{SKINNY-AEAD-M3-Enc}(K, N, A, M)$
In: Key K, nonce N (both 128 bit), associated data A, message M (both arbitrarily long)
Out: (C, tag), where C is the ciphertext with $|C| = |M|$ and tag is a 64-bit tag

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001010 \| N \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001011 \| N \| K}(\mathrm{pad10^*}(A_{\ell_a}))$
//Encryption
$M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m} \leftarrow M$ with $|M_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|M_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $C_i \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001000 \| N \| K}(M_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $M_{\ell_m} = \epsilon$ **then**
  $C_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001100 \| N \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001001 \| N \| K}(0^{128})$
  $C_{\ell_m} \leftarrow M_{\ell_m} \oplus \mathrm{trunc}_{|M_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10^*}(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001101 \| N \| K}(\Sigma)$
$C \leftarrow C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m}$
//Tag generation
$\mathsf{tag} \leftarrow \mathrm{trunc}_{64}(T \oplus \mathsf{Auth})$
return (C, tag)

Algorithm 6 The decryption algorithm $\texttt{SKINNY-AEAD-M3-Dec}(K, N, A, C, \mathsf{tag})$
In: Key K, nonce N (both 128 bit), associated data A, ciphertext C (both arbitrarily long), 64-bit tag tag
Out: M if tag is valid, $\perp$ otherwise

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001010 \| N \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001011 \| N \| K}(\mathrm{pad10}^*(A_{\ell_a}))$
//Decryption
$C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m} \leftarrow C$ with $|C_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|C_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $M_i \leftarrow \texttt{SKINNY-128-384}^{-1}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001000 \| N \| K}(C_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $C_{\ell_m} = \epsilon$ **then**
  $M_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001100 \| N \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001001 \| N \| K}(0^{128})$
  $M_{\ell_m} \leftarrow C_{\ell_m} \oplus \mathrm{trunc}_{|C_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10}^*(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00001101 \| N \| K}(\Sigma)$
$M \leftarrow M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m}$
//Tag verification
$\mathsf{tag}' \leftarrow \mathrm{trunc}_{64}(T \oplus \mathsf{Auth})$
if $\mathsf{tag}' = \mathsf{tag}$ **then**
  return M
else
  return $\perp$

Algorithm 7 The authenticated encryption algorithm $\texttt{SKINNY-AEAD-M4-Enc}(K, N, A, M)$
In: 128-bit key K, 96-bit nonce N, associated data A, message M (both arbitrarily long)
Out: (C, tag), where C is the ciphertext with $|C| = |M|$ and tag is a 64-bit tag

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011010 \| N \| 0^{32} \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011011 \| N \| 0^{32} \| K}(\mathrm{pad10}^*(A_{\ell_a}))$
//Encryption
$M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m} \leftarrow M$ with $|M_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|M_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $C_i \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011000 \| N \| 0^{32} \| K}(M_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $M_{\ell_m} = \epsilon$ **then**
  $C_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011100 \| N \| 0^{32} \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011001 \| N \| 0^{32} \| K}(0^{128})$
  $C_{\ell_m} \leftarrow M_{\ell_m} \oplus \mathrm{trunc}_{|M_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10}^*(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011101 \| N \| 0^{32} \| K}(\Sigma)$
$C \leftarrow C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m}$
//Tag generation
$\mathsf{tag} \leftarrow \mathrm{trunc}_{64}(T \oplus \mathsf{Auth})$
return (C, tag)

Algorithm 8 The decryption algorithm $\texttt{SKINNY-AEAD-M4-Dec}(K, N, A, C, \mathsf{tag})$
In: 128-bit key K, 96-bit nonce N, associated data A, ciphertext C (both arbitrarily long), 64-bit tag tag
Out: M if tag is valid, $\perp$ otherwise

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011010 \| N \| 0^{32} \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011011 \| N \| 0^{32} \| K}(\mathrm{pad10}^*(A_{\ell_a}))$
//Decryption
$C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m} \leftarrow C$ with $|C_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|C_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{63} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $M_i \leftarrow \texttt{SKINNY-128-384}^{-1}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011000 \| N \| 0^{32} \| K}(C_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
if $C_{\ell_m} = \epsilon$ **then**
  $M_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011100 \| N \| 0^{32} \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011001 \| N \| 0^{32} \| K}(0^{128})$
  $M_{\ell_m} \leftarrow C_{\ell_m} \oplus \mathrm{trunc}_{|C_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{64}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10}^*(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-384}_{\mathrm{rev}_{64}(\mathrm{LFSR}) \| 0^{56} \| 00011101 \| N \| 0^{32} \| K}(\Sigma)$
$M \leftarrow M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m}$
//Tag verification
$\mathsf{tag}' \leftarrow \mathrm{trunc}_{64}(T \oplus \mathsf{Auth})$
if $\mathsf{tag}' = \mathsf{tag}$ **then**
  return M
else
  return $\perp$

`SKINNY-AEAD` **with** `SKINNY-128-256`**.** This case applies to the members M5 and M6 (refer to Table 1). It is very similar to the previous case, the main difference being the definition of the tweakey states due to their smaller sizes.

Domain Separation. The domain separation is exactly the same as in the previous case. Note that b_4 is always fixed to 1 as only 96-bit nonces can be used in the members M5 and M6.

Associated Data Processing. The computation for associated data A is very similar to the previous case. The difference is that each associated data block A_i is processed by `SKINNY-128-256` as a plaintext input under a 256-bit tweakey, where the structure of the 256-bit tweakey is as follows.

- The tweakey bytes $tk_0, \ldots, tk_{15}$ store 3 bytes from a 24-bit LFSR, the single byte for the domain separation, followed by the 12-byte nonce N. The byte for the domain separation is fixed to d_2, i.e., $0001t_\ell 010$, for a non-padded block and to $d_3 = 0001t_\ell 011$ for a padded block. The 24-bit LFSR is defined below.
  Let $x_{23}\|x_{22}\|x_{21}\|\cdots\|x_2\|x_1\|x_0$ denote the 24 bits of the LFSR. It is initialized to $LFSR_0 = 0^{23}1$ and updated as $\mathrm{LFSR}_{t+1} = \mathrm{upd}_{24}(\mathrm{LFSR}_t)$, where the update function upd_{24} is defined by the polynomial $x^{24} + x^4 + x^3 + x + 1$ as

$$\mathrm{upd}_{24}\colon x_{23}\|x_{22}\|\ldots\|x_1\|x_0 \mapsto y_{23}\|y_{22}\|\ldots\|y_1\|y_0$$

  with

$$\begin{aligned} y_i &\leftarrow x_{i-1} \text{ for } i \in \{23, 22, \ldots, 1\}\backslash\{4, 3, 1\},\\ y_4 &\leftarrow x_3 \oplus x_{23},\\ y_3 &\leftarrow x_2 \oplus x_{23},\\ y_1 &\leftarrow x_0 \oplus x_{23},\\ y_0 &\leftarrow x_{23}. \end{aligned}$$

  Before loaded in the tweakey state, the order of the bytes of the LFSR state is reversed, i.e., $tk_0\|tk_1\|\ldots\|tk_{15} = \mathrm{rev}_{24}(\mathrm{LFSR}_t)\|d_2\|N$ (resp. $tk_{15} = d_3$ for the last padded block), where rev_{24} is defined by

$$\mathrm{rev}_{24}\colon x_2\|x_1\|x_0 \mapsto x_0\|x_1\|x_2 \quad (\forall i\colon |x_i| = 8).$$

- The tweakey bytes $tk_{16}\|tk_{17}\|\ldots\|tk_{31}$ store the 128-bit key K.

Encryption and Decryption. The encryption of M is also very similar to the previous case. Also, decryption and tag verification is straightforward.

Formally, we provide the algorithms of the authenticated encryption members M5 and M6, together with their decryption and tag verification procedure, in Algorithms 9, 10 and 11, 12, respectively.

Remarks for Further Extension. Here, we explain two additional features of our AEAD schemes, which are not officially included in our submission but can be implemented efficiently depending on the user's demand.

Supporting More than 2^{64} *blocks with* `SKINNY-128-384`*.* Recall that in `SKINNY-128-384`-based members, the tweakey bytes $tk_0, \ldots, tk_{15}$ store 8 bytes for a 64-bit LFSR, followed by 7 bytes of zeros, and then a single byte for the domain separation. If the user wants to support input data of more than 2^{64} blocks, it is possible to replace the 7 bytes

Algorithm 9 The authenticated encryption algorithm $\texttt{SKINNY-AEAD-M5-Enc}(K, N, A, M)$
In: 128-bit key K, 96-bit nonce N, associated data A, message M (both arbitrarily long)
Out: (C, tag), where C is the ciphertext with $|C| = |M|$ and tag is a 128-bit tag

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{23} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010010 \| N \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010011 \| N \| K}(\mathrm{pad10^*}(A_{\ell_a}))$
//Encryption
$M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m} \leftarrow M$ with $|M_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|M_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{23} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $C_i \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010000 \| N \| K}(M_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
if $M_{\ell_m} = \epsilon$ **then**
  $C_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010100 \| N \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010001 \| N \| K}(0^{128})$
  $C_{\ell_m} \leftarrow M_{\ell_m} \oplus \mathrm{trunc}_{|M_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10^*}(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010101 \| N \| K}(\Sigma)$
$C \leftarrow C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m}$
//Tag generation
$\mathsf{tag} \leftarrow T \oplus \mathsf{Auth}$
return (C, tag)

of zeros by the following 56-bit LFSR. Note that this LFSR would be updated every 2^{64} blocks, hence very rarely in comparison to the 64-bit LFSR. Let $x_{55} \| x_{54} \| \cdots \| x_1 \| x_0$ denote the 56 bits of the 56-bit LFSR. It is initialized to $\mathrm{LFSR}_0 = 0^{55} \| 1$ and updated as $\mathrm{LFSR}_{t+1} = \mathrm{upd}_{56}(\mathrm{LFSR}_t)$, where the update function upd_{56} is defined by the polynomial $x^{56} + x^7 + x^4 + x^2 + 1$ as

$$\mathrm{upd}_{56} \colon x_{55} \| x_{54} \| \ldots \| x_1 \| x_0 \quad \longrightarrow \quad y_{55} \| y_{54} \| \ldots \| y_1 \| y_0$$

with:

$$\begin{aligned}
&y_i \leftarrow x_{i-1} \text{ for } i \in \{55, 54, \ldots, 1\} \backslash \{7, 4, 2\}, \\
&y_7 \leftarrow x_6 \oplus x_{55}, \\
&y_4 \leftarrow x_3 \oplus x_{55}, \\
&y_2 \leftarrow x_1 \oplus x_{55}, \\
&y_0 \leftarrow x_{55}.
\end{aligned}$$

We stress that this additional functionality is only available in SKINNY-128-384-based members, and cannot be adopted in SKINNY-128-256-based members.

Algorithm 10 The decryption algorithm $\texttt{SKINNY-AEAD-M5-Dec}(K, N, A, C, \mathsf{tag})$
In: 128-bit key K, 96-bit nonce N, associated data A, ciphertext C (both arbitrarily long), 128-bit tag tag
Out: M if tag is valid, $\perp$ otherwise

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{23} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010010 \| N \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010011 \| N \| K}(\mathrm{pad10^*}(A_{\ell_a}))$
//Decryption
$C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m} \leftarrow C$ with $|C_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|C_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{23} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $M_i \leftarrow \texttt{SKINNY-128-256}^{-1}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010000 \| N \| K}(C_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
if $C_{\ell_m} = \epsilon$ **then**
  $M_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010100 \| N \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010001 \| N \| K}(0^{128})$
  $M_{\ell_m} \leftarrow C_{\ell_m} \oplus \mathrm{trunc}_{|C_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10^*}(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00010101 \| N \| K}(\Sigma)$
$M \leftarrow M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m}$
//Tag verification
$\mathsf{tag}' \leftarrow T \oplus \mathsf{Auth}$
if $\mathsf{tag}' = \mathsf{tag}$ **then**
  return M
else
  return $\perp$

Acceleration of Associated Data Processing. When associated data A is processed, we fix 128 bits and 96 bits of the tweakey state to the nonce value N for `SKINNY-128-384`- and `SKINNY-128-256`-based members. We note that it is not strictly necessary to include N during the associated data processing, hence a potential acceleration of the associated data processing could replace N with bits from A. This would reduce the number of tweakable block cipher calls for processing A. In particular, the number of calls could be halved in `SKINNY-128-384`-based members.

Algorithm 11 The authenticated encryption algorithm $\texttt{SKINNY-AEAD-M6-Enc}(K, N, A, M)$
In: 128-bit key K, 96-bit nonce N, associated data A, message M (both arbitrarily long)
Out: (C, tag), where C is the ciphertext with $|C| = |M|$ and tag is a 64-bit tag

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{23} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011010 \| N \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011011 \| N \| K}(\mathrm{pad10^*}(A_{\ell_a}))$
//Encryption
$M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m} \leftarrow M$ with $|M_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|M_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{23} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $C_i \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011000 \| N \| K}(M_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
if $M_{\ell_m} = \epsilon$ **then**
  $C_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011100 \| N \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011001 \| N \| K}(0^{128})$
  $C_{\ell_m} \leftarrow M_{\ell_m} \oplus \mathrm{trunc}_{|M_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10^*}(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011101 \| N \| K}(\Sigma)$
$C \leftarrow C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m}$
//Tag generation
$\mathsf{tag} \leftarrow \mathrm{trunc}_{64}(T \oplus \mathsf{Auth})$
return (C, tag)

Algorithm 12 The decryption algorithm $\texttt{SKINNY-AEAD-M6-Dec}(K, N, A, C, \mathsf{tag})$
In: 128-bit key K, 96-bit nonce N, associated data A, ciphertext C (both arbitrarily long), 64-bit tag tag
Out: M if tag is valid, $\perp$ otherwise

//Associated data processing
$A_0 \| A_1 \| \ldots \| A_{\ell_a - 1} \| A_{\ell_a} \leftarrow A$ with $|A_i| = 128$ for $i \in \{0, \ldots, \ell_a - 1\}$ and $|A_{\ell_a}| < 128$
$\mathsf{Auth} \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{23} \| 1$
for all $i = 0, \ldots, \ell_a - 1$ **do**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011010 \| N \| K}(A_i)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
if $A_{\ell_a} \neq \epsilon$ **then**
  $\mathsf{Auth} \leftarrow \mathsf{Auth} \oplus \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011011 \| N \| K}(\mathrm{pad10}^*(A_{\ell_a}))$
//Decryption
$C_0 \| C_1 \| \ldots \| C_{\ell_m - 1} \| C_{\ell_m} \leftarrow C$ with $|C_i| = 128$ for $i \in \{0, \ldots, \ell_m - 1\}$ and $|C_{\ell_m}| < 128$
$\Sigma \leftarrow 0^{128}$
$\mathrm{LFSR} \leftarrow 0^{23} \| 1$
for all $i = 0, \ldots, \ell_m - 1$ **do**
  $M_i \leftarrow \texttt{SKINNY-128-256}^{-1}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011000 \| N \| K}(C_i)$
  $\Sigma \leftarrow \Sigma \oplus M_i$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
if $C_{\ell_m} = \epsilon$ **then**
  $M_{\ell_m} \leftarrow \epsilon$
  $T \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011100 \| N \| K}(\Sigma)$
else
  $R \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011001 \| N \| K}(0^{128})$
  $M_{\ell_m} \leftarrow C_{\ell_m} \oplus \mathrm{trunc}_{|C_{\ell_m}|}(R)$
  $\mathrm{LFSR} \leftarrow \mathrm{upd}_{24}(\mathrm{LFSR})$
  $\Sigma \leftarrow \Sigma \oplus \mathrm{pad10}^*(M_{\ell_m})$
  $T \leftarrow \texttt{SKINNY-128-256}_{\mathrm{rev}_{24}(\mathrm{LFSR}) \| 00011101 \| N \| K}(\Sigma)$
$M \leftarrow M_0 \| M_1 \| \ldots \| M_{\ell_m - 1} \| M_{\ell_m}$
//Tag verification
$\mathsf{tag}' \leftarrow \mathrm{trunc}_{64}(T \oplus \mathsf{Auth})$
if $\mathsf{tag}' = \mathsf{tag}$ **then**
  return M
else
  return $\perp$

2.5 The Hash Functionality SKINNY-Hash

Overall, the SKINNY-Hash family are function-based sponge constructions SKINNY-tk3-Hash and SKINNY-tk2-Hash, in which underlying functions are built with SKINNY-128-384 and SKINNY-128-256, respectively. We recall here that the sponge construction [11] can be based on a cryptographic function as well as a cryptographic permutation.

F_{384}: 384-bit to 384-bit function. We build a function $F_{384} : \{0,1\}^{384} \rightarrow \{0,1\}^{384}$ based on SKINNY-128-384. Let $x \in \{0,1\}^{384}$ be an input to F_{384}. Let $\texttt{SKINNY-128-384}_{tk}(P)$ be the encryption of a plaintext P under a tweakey tk with the SKINNY-128-384 algorithm. The output of F_{384} is computed as follows (see also Figure 8):

$$F_{384}(x) = \texttt{SKINNY-128-384}_x(0^{128}) \Big\| \texttt{SKINNY-128-384}_x(0^7 \| 1 \| 0^{120}) \Big\| \texttt{SKINNY-128-384}_x(0^6 \| 1 \| 0^{121}).$$

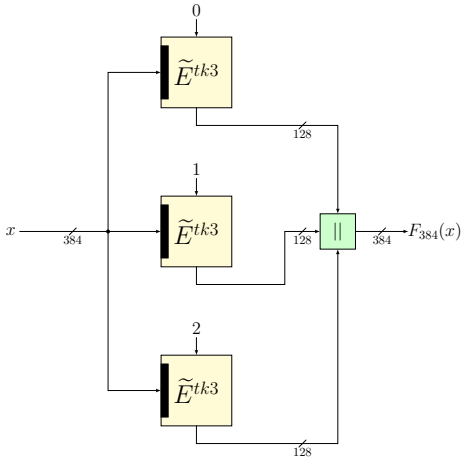


Figure 8: Construction of the function F_{384} used in SKINNY-tk3-Hash.

F_{256}: 256-bit to 256-bit function. We build a function $F_{256} : \{0,1\}^{256} \rightarrow \{0,1\}^{256}$ based on SKINNY-128-256. Let $\texttt{SKINNY-128-256}_{tk}(P)$ be the encryption of a plaintext P under a tweakey tk with the SKINNY-128-256 algorithm. The output of F_{256} is computed as follows (see also Figure 9):

$$F_{256}(x) = \texttt{SKINNY-128-256}_x(0^{128}) \Big\| \texttt{SKINNY-128-256}_x(0^7 \| 1 \| 0^{120}).$$

SKINNY-tk3-Hash. The computation of SKINNY-tk3-Hash simply follows the well-known sponge construction. Differently from many of existing instantiations, we use the *function* F_{384} as an underlying primitive. The construction is illustrated in Figure 10.

The 384-bit state, S_{384}, is divided into 128-bit rate and 256-bit capacity, which are initialized to the following values:

$$\begin{aligned} {}_rIV_{384} &= 0^{128}, \\ {}_cIV_{384} &= 10^{255}, \\ S_{384} &= {}_rIV_{384} \,\|\, {}_cIV_{384}. \end{aligned}$$

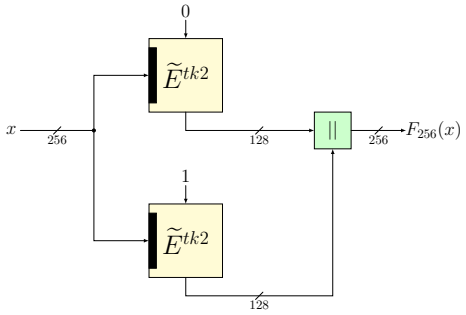


Figure 9: Construction of F_{256}.

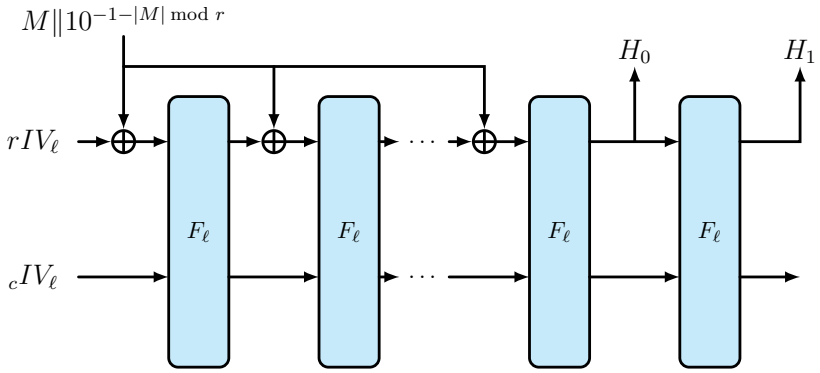


Figure 10: The structure of `SKINNY-Hash` based on a sponge.

The padding pad10* is applied to an input message M (note that the padding is always applied, even if $|M|$ is already a multiple of 128). The message blocks M_i are XORed to the rate part of the state during the absorbing phase.

After the absorbing phase, the 128 bits of the rate are extracted as the first 128 bits of the 256-bit digest. Then, $S_{384} \leftarrow F_{384}(S_{384})$ is applied once again and the 128 bits of the rate are extracted as the last 128 bits of the 256-bit digest. The formal algorithm is specified in Algorithm 13.

`SKINNY-tk2-Hash`. The 256-bit state S_{256}, is divided into a 32-bit rate part and a 224-bit capacity part, which are initialized to the following values:

$$\begin{aligned} {}_rIV_{256} &= 0^{32}, \\ {}_cIV_{256} &= 10^{223}, \\ S_{256} &= {}_rIV_{256}\|{}_cIV_{256}. \end{aligned}$$

A difference with the previous case is that the message M now has to be padded such that its length is a multiple of 32 bits. Therefore, we apply the padding function pad10*_{32} which is defined as

$$\text{pad10*}_{32}\colon X \mapsto X\|1\|0^{31-|X| \bmod 32}.$$

The message blocks M_i are XORed to the rate part of the state during the absorbing phase. After the absorbing phase, the first 128 bits of the state are extracted as the first 128 bits of the 256-bit digest. Then, $S_{256} \leftarrow F_{256}(S_{256})$ is applied once again and the first 128 bits of the state are extracted as the last 128 bits of the 256-bit digest. This means

Algorithm 13 The hashing algorithm `SKINNY-tk3-Hash`
In: Message M of arbitrary length
Out: 256-bit digest H

//Absorbing phase
$M_0 \| M_1 \| \dots \| M_{\ell_m - 1} \leftarrow \text{pad10*}(M)$ with $|M_i| = 128$ for $i \in \{0, \dots, \ell_m - 1\}$
$S_{384} \leftarrow 0^{128} \| 1 \| 0^{255}$
for all $i = 0, \dots, \ell_m - 1$ **do**
    $S_{384} \leftarrow F_{384}\big(S_{384} \oplus (M_i \| 0^{256})\big)$
//Squeezing phase
$H_0 \leftarrow \text{trunc}_{128}(S_{384})$
$S_{384} \leftarrow F_{384}\big(S_{384}\big)$
$H_1 \leftarrow \text{trunc}_{128}(S_{384})$
$H \leftarrow H_0 \| H_1$
return H

that in the squeezing phase, the rate is extended to 128 bits and the capacity is reduced to 128 bits. The formal algorithm is specified in Algorithm 14.

Algorithm 14 The hashing algorithm `SKINNY-tk2-Hash`
In: Message M of arbitrary length
Out: 256-bit digest H

//Absorbing phase
$M_0 \| M_1 \| \dots \| M_{\ell_m - 1} \leftarrow \text{pad10*}_{32}(M)$ with $|M_i| = 32$ for $i \in \{0, \dots, \ell_m - 1\}$
$S_{256} \leftarrow 0^{32} \| 1 \| 0^{223}$
for all $i = 0, \dots, \ell_m - 1$ **do**
    $S_{256} \leftarrow F_{256}\big(S_{256} \oplus (M_i \| 0^{224})\big)$
//Squeezing phase
$H_0 \leftarrow \text{trunc}_{128}(S_{256})$
$S_{256} \leftarrow F_{256}\big(S_{256}\big)$
$H_1 \leftarrow \text{trunc}_{128}(S_{256})$
$H \leftarrow H_0 \| H_1$
return H

Table of Parameters and Security of `SKINNY-Hash`. For a summary, parameters of `SKINNY-Hash` are listed in Table 4.

Table 4: Parameters for `SKINNY-Hash`. The number of blocks of the first preimage is denoted by L.

Algorithm	State size	Absorb		Squeeze		Security	Security
		Rate	Capacity	Rate	Capacity	(collision)	(2nd preimage)
`SKINNY-tk3-Hash`	384	128	256	128	256	128	$256 - \log_2(L)$
`SKINNY-tk2-Hash`	256	32	224	128	128	112	$224 - \log_2(L)$

3 Security Claims

We provide our security claims for the different variants of SKINNY-AEAD and SKINNY-Hash in Table 5. Basically, for all versions of SKINNY-AEAD, we claim full 128-bit security for key recovery, confidentiality and integrity (unless the tag size is smaller than 128 bits, in which case the integrity security claims drop to the tag size) in the *nonce-respecting model*. For all versions of SKINNY-Hash, we claim that it is hard to find a collision, preimage or second-preimage with substantially less than $2^{c/2}$ hash evaluations, where c represents the capacity bitsize ($c = 256$ for M5 and $c = 224$ for M6).

One can see that we do claim full 128-bit security for all variants of SKINNY-AEAD with a tag size of 128 bit for a nonce-respecting user. More precisely, confidentiality is perfectly guaranteed and the forgery probability is $2^{-\tau}$, where τ denotes the tag size, independently of the number of blocks of data in encryption/decryption queries made by the adversary. This is very different than other modes like AES-GCM [32] or OCB3 [27], which only ensure birthday-bound security. In comparison, OCB3 only provides security up to the birthday bound, more precisely up to roughly $2^{n/2}$ *blocks of data* since it relies on XE/XEX (a construction of a tweakable block cipher from a standard block cipher with security only up to the birthday bound). To give an numerical example, with 2^{40} blocks ciphered (about 16 TeraBytes), one gets an advantage of about 2^{-48} to generate a valid tag for most operating modes in the nonce-respecting scenario. For the same amount of data, the advantage remains 2^{-128} for members M1/M2/M5 of SKINNY-AEAD.

	Security (bits)					
SKINNY-AEAD (nonce-respecting)	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**
Key recovery	128	128	128	128	128	128
Confidentiality for the plaintext	128	128	128	128	128	128
Integrity for the plaintext/AD/nonce	128	128	64	64	128	64

	Security (bits)	
SKINNY-Hash	**M1/M2/M3/M4**	**M5/M6**
Collision	128	112
(2nd)-preimage	128	112

Table 5: Security claims of SKINNY-AEAD and SKINNY-Hash. The bit security of our designs is expressed in terms of calls to the internal primitive, up to a small logarithmic factor.

We assume that the total size of the associated data and the total size of the message in SKINNY-AEAD do not exceed 2^{68} bytes for M1/M2/M3/M4 and 2^{28} bytes for M5/M6. Moreover, the maximum number of messages that can be handled for a same key is 2^{n_l} for all variants of SKINNY-AEAD ($n_l = 128$ for M1/M3, $n_l = 96$ for M2/M4/M5/M6). This will ensure that as long as different fixed-length nonces are used, the tweak inputs of all the tweakable block cipher calls are all unique.

Related-Cipher Attacks. By encoding the the length of the tag and nonce into the domain separation, we obtain a proper separation between the SKINNY-AEAD members that employ the same instance of the SKINNY tweakable block cipher. We do not claim security against related-cipher attacks between members that employ the two different instances SKINNY-128-384 and SKINNY-128-256, e.g., M2 and M5.

Nonce-Misuse Setting. The above security claims are void under reuse of nonces. As pointed out in [47] for the case of `Deoxys-I`, the scheme is vulnerable to a universal forgery attack and a CCA decryption attack with complexity of only three queries. Because we are basically using the same mode, the attacks would apply to `SKINNY-AEAD` as well.

4 Design Rationale

4.1 Rationale for the Tweakable Block Ciphers

In this section, we briefly recall the design rationale for the tweakable block ciphers SKINNY-128-256 and SKINNY-128-384 as given in [7, 8].

Most importantly, we decided *not* to modify the cipher from its original specification. The rationale for this is that none of the extensive third-party cryptanalysis, that we discuss in detail in Section 5, pointed to any weakness of the cipher nor any bad design choices. Indeed, all the third-party cryptanalysis confirmed the validity of the original design and its rationale. We furthermore do not see any change in the specification that would improve the cipher to the extent that would justify such a modification. All design choices of SKINNY are optimized for its goal: One cipher well suited for many lightweight applications.

When designing SKINNY, one of the main criteria was to only add components which are vital for the security of the primitive, removing any unnecessary operation. The construction of SKINNY has been done through several iterations, approaching the exact spot where good performance meets strong security arguments. We detail in this section how we tried to follow this direction for each layer of the cipher.

We define the adversarial model **SK** (resp. **TK1**, **TK2** or **TK3**) where the attacker cannot (resp. can) introduce differences in the tweakey state(s).

General Design and Components Rationale. A first and important decision was to choose between a Substitution-Permutation Network (SPN), or a Feistel network. We started from a SPN construction as it is generally easier to provide stronger bounds on the number of active Sboxes. However, we note that there is a dual bit-sliced view of SKINNY that resembles some generalized Feistel network. Somehow, one can view the cipher as a primitive in between an SPN and an "AND-rotation-XOR" function like SIMON. We try to get the best of both worlds by benefiting the nice implementation tradeoffs of the latter, while organizing the state in an SPN view so that bounds on the number of active Sboxes can be easily obtained.

The absence of whitening key is justified by the reduction of the control logic: by always keeping the exact same round during the entire encryption process we avoid the control logic induced by having a last non-repeating layer at the end of the cipher. Besides, this simplifies the general description and implementation of the primitive. Obviously, having no whitening key means that a few operations of the cipher have no impact on the security. This is actually the case for both the beginning and the end of the ciphering process in SKINNY since the key addition is done in the middle of the round, with only half of the state being involved with this key addition every round.

A crucial feature of SKINNY is the easy generation of several tweakey size versions, while keeping the general structure and most of the security analysis untouched. Using bigger tweakey material is done by following the STK construction [21].

SubCells. The choice of the Sbox is obviously a crucial decision in an SPN cipher and we have spent a lot of efforts on looking for the best possible candidate.

As the entire search space for an 8 bit Sbox is far from being completely searchable, we considered a subclass of the entire search space: we have tested all possible Sboxes built by iterating several times a NOR/XOR combination and a bit permutation. The final 8-bit Sbox candidate $\mathcal{S}_8$ provides a good tradeoff between security and area cost. It has maximal differential transition probability of 2^{-2}, maximal absolute linear bias of 2^{-2}, and algebraic degree 6.

Note that $\mathcal{S}_8$ has the interesting feature that their inverse is computed almost identically to the forward direction (as they are based on a generalized Feistel structure) and with

exactly the same number of operations. Thus, our design reasoning also holds when considering the decryption process.

`AddConstants.` The constants in `SKINNY` have several goals: differentiate the rounds, differentiate the columns and avoid symmetries, complicate subspace cryptanalysis and complicated attacks exploiting fixed points from the Sbox. In order to differentiate the rounds, we simply need a counter, and since the number of rounds of all `SKINNY` versions is smaller than 64, the most hardware friendly solution is to use a very cheap 6-bit affine LFSR (like in `LED` [18]) that requires only a single XNOR gate per update. The 6 bits are then dispatched to the two first rows of the first column (this will maximize the constants spread after the `ShiftRows` and `MixColumns`), which will already break the symmetry between columns.

In order to avoid symmetries, fixed points and more generally subspaces to spread, we need to introduce different constants in several cells of the internal state. The round counter will already naturally have this goal, yet, in order to increase that effect, we have added a "`1`" bit to the third row, which is almost free in terms of implementation cost. This will ensure that symmetries and subspaces are broken even more quickly, and in particular independently of the round counter.

`AddRoundTweakey.` The tweakey schedule of `SKINNY` follows closely the `STK` construction from [21] that allows to easily get bounds on the number of active Sboxes in the related-tweakey model. Yet, we have changed a few parts. First, instead of using multiplications by 2 and 3 in a finite field, we have instead replaced these tweakey cells updates by cheap 8-bit LFSRs to minimize the hardware cost. All our LFSRs require only a single XOR for the update, and we have checked that the differential cancellation behavior of these interconnected LFSRs is as required by the `STK` construction: for a given position, a single cancellation can only happen every 15 rounds for **TK2**, and same with two cancellations for **TK3**.

Another important generalization of the `STK` construction is the fact that every round we XOR only half of the internal cipher state with some subtweakey. The goal was to optimize hardware performances of `SKINNY`, and it actually saves an important amount of XORs in a round-based implementation, while the security bounds remain strong. Another advantage is that we can now update the tweakey cells only before they are incorporated to the cipher internal state. Thus, half of tweakey cells only will be updated every round and the period of the cancellations naturally doubles: for a certain cell position, a single cancellation can only happen every 30 rounds for **TK2** and two cancellations can only happen every 30 rounds for **TK3**.

The tweakey permutation P_T has been chosen to maximize the bounds on the number of active Sboxes that we could obtain in the related-tweakey model (note that it has no impact in the single-key model). Besides, we have enforced for P_T the special property that all cells located in third and fourth rows are sent to the first and second rows, and vice-versa. Since only the first and second rows of the tweakey states are XORed to the internal state of the cipher, this ensures that both halves of the tweakey states will be equally mixed to the cipher internal state (otherwise, some tweakey bytes might be more involved in the ciphering process than others). Finally, the cells that will not be directly XORed to the cipher internal state can be left at the same relative position. On top of that, we only considered those variants of P_T that consist of a single cycle.

We note that since the cells of the first tweakey word $TK1$ are never updated, they can be directly hardwired to save some area if the situation allows.

`ShiftRows and MixColumns.` Due to its strong sparseness, `SKINNY`'s binary diffusion matrix **M** has only a differential or linear branching number of two. In order to compensate

for the small branch number, we designed $\mathbf{M}$ such that when a branching two differential transition occurs, the next round will likely lead to a much higher branching number. Looking at $\mathbf{M}$, the only way to meet branching two is to have an input difference in either the second or the fourth input only. This leads to an input difference in the first or third element for the next round, which then diffuses to many output elements. The differential characteristic with a single active Sbox per round is therefore impossible, and actually we are able to prove at least 96 active Sboxes for 20 rounds. Thus, for the very cheap price of a differential branching two binary diffusion matrix, we are in fact getting a better security than expected when looking at the iteration of several rounds. The effect is the same with linear branching (for which we only need to look at the transpose of the inverse of $\mathbf{M}$, i.e. $(\mathbf{M}^{-1})^{\top}$).

We have considered all possibilites for $\mathbf{M}$ that can be implemented with at most three XOR operations and eventually kept the `MixColumns` matrices that, in combination with `ShiftRows`, guaranteed high diffusion and led to strong bounds on the minimal number of active Sboxes in the single-key model.

Note that another important criterion came into play regarding the choice of the diffusion layer of `SKINNY`: it is important that the key material impacts as fast as possible the cipher internal state. This is in particular a crucial point for `SKINNY` as only half of the state is mixed with some key material every round, and since there is no whitening keys. Besides, having a fast key diffusion will reduce the impact of meet-in-the-middle attacks. Once the two first rows of the state were arbitrarily chosen to receive the key material, given a certain subtweakey, we could check how many rounds were required (in both encryption and decryption directions) to ensure that the entire cipher state depends on this subtweakey. Our final choice of `MixColumns` is optimal: only a single round is required in both forward and backward directions to ensure this diffusion.

Our entire design has been crafted to allow good provable bounds on the minimal number of differential or linear active Sboxes, not only for the single-key model, but also in the related-key model (or more precisely the related-tweakey model in our case). Those bounds are given in Table 7 in Section 5.1.

Finally, in terms of diffusion, all versions of `SKINNY` achieve full diffusion after only 6 rounds (forwards or backwards), while `SIMON` versions with 64-bit block size requires 9 rounds, and even 13 rounds for `SIMON` versions with 128-bit block size [25] (`AES-128` reaches full diffusion after 2 of its 10 rounds). Again, the diffusion comparison according to the total number of rounds is at `SKINNY`'s advantage.

4.2 Rationale for the AEAD scheme

The reason for choosing the `ΘCB3` mode for the tweakable block cipher `SKINNY-128-384` or `SKINNY-128-256` is its provable security providing *full* security in the nonce-respecting setting. More precisely, for `ΘCB3` using an ideal tweakable block cipher, confidentiality is perfectly guaranteed and the forgery probability is independent of the number of blocks of data in encryption/decryption queries made by the adversary. Those strong security guarantees along with its performance features are the design rationale for our choice.

We state the security bound of `ΘCB3` in the nonce-respecting setting:

Lemma 2 of [27]. *Let $\prod = \Theta CB3[\tilde{E}, \tau]$ where $\tilde{E}$ is an ideal tweakable block cipher. Let $\mathcal{A}$ be an adversary. Then $\boldsymbol{Adv}^{priv}_{\prod}(\mathcal{A}) = 0$ and $\boldsymbol{Adv}^{auth}_{\prod}(\mathcal{A}) \leq (2^{n-\tau})/(2^n - 1)$.*

We denote by $\mathbf{Adv}^{\pm\mathrm{prp}}_{\texttt{SKINNY-TK2}}(\mathcal{A})$ and $\mathbf{Adv}^{\pm\mathrm{prp}}_{\texttt{SKINNY-TK3}}(\mathcal{A})$ the PRP-advantage against `SKINNY-128-256` and `SKINNY-128-384` respectively. Replacing the ideal tweakable block cipher with `SKINNY`, we have the security bounds for our members as shown in Table 6.

Table 6: Provable security bounds for our provided AEAD members.

Members	Security Bounds
M1, M2	$\mathbf{Adv}^{\mathrm{priv}}_{\prod}(\mathcal{A}) \leq \mathbf{Adv}^{\pm\mathrm{prp}}_{\texttt{SKINNY-TK3}}(\mathcal{A})$ $\mathbf{Adv}^{\mathrm{auth}}_{\prod}(\mathcal{A}) \leq (2^{128}-1)^{-1} + \mathbf{Adv}^{\pm\mathrm{prp}}_{\texttt{SKINNY-TK3}}(\mathcal{A})$
M3, M4	$\mathbf{Adv}^{\mathrm{priv}}_{\prod}(\mathcal{A}) \leq \mathbf{Adv}^{\pm\mathrm{prp}}_{\texttt{SKINNY-TK3}}(\mathcal{A})$ $\mathbf{Adv}^{\mathrm{auth}}_{\prod}(\mathcal{A}) \leq (2^{64}-2^{-64})^{-1} + \mathbf{Adv}^{\pm\mathrm{prp}}_{\texttt{SKINNY-TK3}}(\mathcal{A})$
M5	$\mathbf{Adv}^{\mathrm{priv}}_{\prod}(\mathcal{A}) \leq \mathbf{Adv}^{\pm\mathrm{prp}}_{\texttt{SKINNY-TK2}}(\mathcal{A})$ $\mathbf{Adv}^{\mathrm{auth}}_{\prod}(\mathcal{A}) \leq (2^{128}-1)^{-1} + \mathbf{Adv}^{\pm\mathrm{prp}}_{\texttt{SKINNY-TK2}}(\mathcal{A})$
M6	$\mathbf{Adv}^{\mathrm{priv}}_{\prod}(\mathcal{A}) \leq \mathbf{Adv}^{\pm\mathrm{prp}}_{\texttt{SKINNY-TK2}}(\mathcal{A})$ $\mathbf{Adv}^{\mathrm{auth}}_{\prod}(\mathcal{A}) \leq (2^{64}-2^{-64})^{-1} + \mathbf{Adv}^{\pm\mathrm{prp}}_{\texttt{SKINNY-TK2}}(\mathcal{A})$

On OCB and Tweakable Block Ciphers. The OCB mode was first published in [38] (i.e., OCB1). It has later been refined to OCB2 in [37] and finally to OCB3 in [27]. That last paper describes the actual ΘCB3 framework we employ in SKINNY-AEAD by using a dedicated tweakable block cipher. The classical OCB (1–3) mode does not employ a dedicated tweakable block cipher, but rather a usual block cipher in an XEX-like construction. Recently, OCB3 employed with the AES was selected as one of the winners of the CAESAR competition in the category for high-performance applications [28]. However, this scheme only offers birthday-bound security.

More generally, a *tweakable block cipher* can be described as a family of block ciphers parameterized by a public parameter, the *tweak*. The idea of a block cipher that gets a public parameter for achieving variability goes back to the design of the Hasty Pudding Cipher [41], a submission to the AES competition. This was later formalized in the notion of a tweakable block cipher by Liskov, Rivest and Wagner at CRYPTO 2002 [30]. The motivation is that independent block cipher calls are needed at the mode-of-operation level, as in OCB. Liskov, Rivest and Wagner suggested that the source of variability should be directly incorporated in the primitive itself instead at the mode-of-operation level. This is a big difference to the classical OCB mode. There, a block cipher E is employed in a construction that can be understood as a tweakable block cipher (i.e., the tweakable block cipher E^T_K is just defined as $E^{(T_1,T_2)}_K(x) = E_K(x \oplus T_1) \oplus T_2)$). In that sense, OCB can be seen as an instance of the more general TAE mode, the *tweakable authenticated encryption mode* defined in [30]. Indeed, Liskov, Rivest and Wagner have already proven a similar statement as Lemma 2 in [27]:

Theorem 3 of [30]. *If $\widetilde{E}$ is a secure tweakable block cipher, then $\widetilde{E}$ used in* TAE *mode will be unforgeable and pseudorandom.*

In other words: The advantage of the adversary only comes from the distinguishing advantage of the tweakable block cipher and not from the mode.

However, the XEX construction used in OCB and also in ΘCB3 does not lead to an ideal tweakable block cipher. In fact, it only offers security up to the *birthday bound*. The TWEAKEY framework [21] was introduced at ASIACRYPT 2014 as a method to build

tweakable block ciphers from scatch (i.e., without employing an already existing underlying block cipher in a specific construction) with strong security arguments against differential and linear attacks. The intention of the TWEAKEY framework was to obtain beyond birthay-bound secure tweakable block ciphers and to consider key and tweak as the similar type of input (called the tweakey) such that the separation into key and tweak can be done by the user in a flexible way.

It is natural to employ a beyond-birthday secure tweakable block cipher in a mode following the `TAE` (resp., `ΘCB3`) framework in order to exploit its full strength. The third-round CAESAR candidate `Deoxys-I` [22] is an already existing example following this design principle.

Our Modifications. In comparison to other modes of operation, we have decided to replace the usual block counter by an LFSR, which can be implemented with just a few operations. There is indeed no reason to use the increment function $x \mapsto x + 1$ over the integers, as the security simply relies on the function having a maximal cycle. The same argument has been made for instance in the original `OCB` mode where Gray codes have been suggested to derive inner tweak values. Here in our AEAD mode, we adopted LFSRs with maximal periods and which can be easily implemented in both hardware and software as block counters.

4.3 Rationale for the Hash Function Scheme

We use the well-known sponge construction, originally presented in [10], that is also adopted in the NIST standard `SHA-3` [15] so that `SKINNY-Hash` can inherit its elegant features. Here, we give some arguments for our design choices with respect to the following points:

1. the sponge construction using a cryptographic function as a building block,
2. the sizes of rate and capacity, and
3. our constructions of the 256- and 384-bit functions.

Function-Based Sponge. Although a lot of existing designs following the sponge framework use a cryptographic permutation as an underlying primitive, the designers do not restrict the underlying primitive to be a permutation and show a lot of analysis for the case that the underlying primitive is a function (see [11] for a detailed documentation on cryptographic sponges and several of its variants). There does not exist any significant disadvantage to base an entire construction on a function instead of a permutation. For example, the bounds for the indifferentiability and the collision resistance are almost identical between those two constructions.

In some case, the function-based sponge constructions is more difficult to attack than the permutation based sponge constructions, because the adversary does not have access to the inverse oracle for the function based constructions. This makes a significant difference of the security against second-preimage attacks. For permutation-based constructions, second preimages can be found by generating collisions on the inner part between queries to f and f^{-1}, which allows a generic attack with a cost of $2^{c/2}$. For function-based designs on the other hand, the best strategy is performing a similar second-preimage attack against Merkle-Damgård constructions [23] that requires $(2^c)/L$ where L is the number of blocks included in the first preimage.

Choices of Rate and Capacity. We adopt the most natural choice for `SKINNY-tk3-Hash`. The 256-bit capacity ensures 128-bit indifferentiability. Hence, no particular attack can be performed under 2^{128} computational cost.

The choice for SKINNY-tk2-Hash is very optimized for lightweight use-cases. The 224-bit capacity in the absorption phase ensures the minimum requirement of 112-bit security. We change the rate and capacity for the squeezing phase to reduce the number of function calls in the squeezing phase. The security in this situation is analyzed in [35]. Let c and c' be the capacity in the absorption and the squeezing phases, respectively. It was shown that c' can be enlarged with preserving $O(2^{c/2})$ security for indifferentiability as long as $c' \geq c/2 + \log_2 c$. We are aiming at 112-bit security, hence the suitable size for c' is $c' \geq 224/2 + \log_2 224 \approx 119.8$. Because we cannot produce 256-bit hash digest in a single block, we set $c' = 128$ so that the 256-bit hash digest can be produced with two blocks.

The results in [35] are for permutation-based schemes, however we confirmed that almost the same bound can be obtained for the function-based schemes. Strictly speaking, the bound is slightly better for the function-based schemes because the adversary cannot access to the inverse oracle.

Rationale of F_{256} and F_{384}. The function F_{256} is indifferentiable from a 256-bit random function up to $O(2^{128})$ queries. Very intuitively, the only way to indifferentiate F_{256} from an ideal object is to find the case that two simulators of $\tilde{E}^{tk2}$ in the ideal world, one is for the plaintext 0 and the other is for the plaintext 1, return the same output value under the same tweakey input. This occurs with probability 2^{-128}.

The same intuitive argument applies to F_{384}. However, the bound is worse than the one for F_{256} by a factor of 3 because the adversary now has three ways to indifferentiate the real and ideal worlds: collision of the simulators output between the first and the second simulators, between the first and third simulators, and between the second and the third simulators.

5 Security Analysis of the SKINNY Tweakable Block Cipher

We claim security of the SKINNY family of tweakable block ciphers in the *related-tweakey model*. We now provide an analysis of its security and then mention the best cryptanalytic results published to date.

5.1 Differential/Linear Cryptanalysis

In order to argue for the resistance of SKINNY against differential and linear attacks, we computed lower bounds on the minimum number of active Sboxes, both in the single-key and related-tweakey models. We recall that, in a differential (resp. linear) characteristic, an Sbox is called *active* if it contains a non-zero input difference (resp. input mask). In contrast to the single-key model, an attacker is allowed to introduce differences (resp. masks) within the tweakey state in the related-tweakey model. We considered the three cases of choosing input differences in **TK1** only, both **TK1** and **TK2**, and in all of the tweakey states **TK1**, **TK2** and **TK3**, respectively. Table 7 presents lower bounds on the number of active Sboxes for up to 30 rounds. For computing the bounds, we generated a Mixed-Integer Linear Programming (MILP) model following the approach in [34, 45].

For lower bounding the number of linear active Sboxes we used the same approach by considering the inverse of the transposed linear transformation, i.e., $\mathbf{M}^{\top}$. However, for the linear case, we only considered the single-key model as there is no cancellation of active Sboxes in linear characteristics, see [26]. Note that those bounds are for single characteristic only and do not quantify any potential clustering into differentials (resp. linear hulls).

Table 7: Lowerbounds on the number of active Sboxes in SKINNY. Note that the bounds on the number of *linear* active Sboxes in the single-key model are also valid in the related-tweakey model. In case the MILP optimization was too long, we provide upper bounds between parentheses. The bounds indicated by $\star$ were obtained from [2], in which the authors used Matsui's algorithm for obtaining the minimum number of active Sboxes.

Model	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
SK	1	2	5	8	12	16	26	36	41	46	51	55	58	61	66
TK1	0	0	1	2	3	6	10	13	16	23	32	38	41	45	49
TK2	0	0	0	0	1	2	3	6	9	12	16	21	25	31	35
TK3	0	0	0	0	0	0	1	2	3	6	10	13	16	19	24
SK Lin	1	2	5	8	13	19	25	32	38	43	48	52	55	58	64

Model	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**
SK	75	82	88	92	96	102	108	112$\star$	116$\star$	124$\star$	128$\star$	132$\star$	136$\star$	142$\star$	148$\star$
TK1	54	59	62	66	70	75	79	83	85	88	95	102	(108)	(112)	(120)
TK2	40	43	47	52	57	59	64	67	72	75	82	85	88	92	96
TK3	27	31	35	43	45	48	51	55	58	60	65	72	77	81	85
SK Lin	70	76	80	85	90	96	102	107	(110)	(118)	(122)	(128)	(136)	(141)	(143)

5.2 Other Attacks

In the original design document [7,8], we also analyzed the security of SKINNY with regard to meet-in-the-middle attacks, impossible differential attacks, integral attacks, slide attacks, invariant subspace cryptanalysis, and algebraic attacks. In this document, we provide a brief summary of the results. For details, we refer the reader to the original documents.

Meet-in-the-Middle Attacks We used the property that full diffusion is achieved after six rounds (both in forward and backward direction) to estimate that meet-in-the middle attacks might work up to at most 22 rounds.

Impossible Differential Attacks We constructed an 11-round truncated impossible differential which can be used for a 16-round key-recovery attack on SKINNY-128 with data, time, and memory complexities of $2^{88.5}$ in the single-key model.

Integral Attacks We constructed a 10-round integral distinguisher and used it for a 14-round key-recovery attack.

Slide Attacks The distinction between the rounds is ensured by the round constants and thus the straightforward slide attacks cannot be applied. However, due to the small state of the LFSR, round constants can collide in different rounds.

We took into account all possible sliding numbers of rounds and deduced what is the difference in the constants that is obtained every time. As these constant differences might impact the best differential characteristic, we experimentally checked the lower bounds on the number of active Sboxes for all these constant differences by using MILP.

In the single-key setting, by allowing any starting round for each value of the slid pair, the lower bounds on the number of active Sboxes reach 36 after 11 rounds, and 41 after 12 rounds. We thus expect that slide attacks do not threaten the security of SKINNY.

Invariant Subspace Attacks The non-trivial key schedule already provides a good protection against such attacks for a larger number of rounds. The main concern that remains are large-dimensional subspaces that propagate invariant through the Sbox. We checked that no such invariant subspaces exist. Moreover, we computed all affine subspaces of dimension larger than two that get mapped to (different) affine subspaces and checked if those can be chained to what could be coined a *subspace characteristic*. It turns out that those subspaces can be chained only for a very small number of rounds. To conclude, the non-trivial key schedule and the use of round-constants seem to sufficiently protect `SKINNY` against those attacks.

Algebraic Attacks The Sbox $\mathcal{S}_8$ of `SKINNY-128` has an algebraic degree of 6 and thus, algebraic attacks do not seem to be a threat.

5.3 Third-Party Cryptanalysis

Since the publication of the cipher in 2016, there has been lots of cryptanalysis by external researchers. To the best of our knowledge, we provide a complete list of published results related to (mathematical) cryptanalysis of `SKINNY`, as of December 2018. We found 20 such papers in total.

Twelve of those, namely [3, 4, 5, 16, 19, 36, 40, 44, 48, 49, 50, 51], only consider the variant of `SKINNY` with a block size of 64 bit and we do not mention their results here. We briefly mention the results of the remaining 8 papers in the following.

In [31], the authors conduct cryptanalysis on various variants of `SKINNY` in the related-tweakey model. For `SKINNY-128-256` (resp. `SKINNY-128-384`), they obtain a 23-round (resp. 27-round) related-tweakey impossible differential attack with time complexity $2^{251.47}$ (resp. 2^{378}), data of $2^{124.47}$ (resp. $2^{126.03}$) chosen plaintexts and 2^{248} (resp. 2^{368}) memory. The impossible differential attack uses a truncated related-tweakey impossible differential over 12 rounds (resp. 16 rounds for the impossible differential attack on `SKINNY-128-384`). Those complexities were improved under the assumption that the public tweak is loaded in TK-1. For `SKINNY-128-384` (resp. `SKINNY-128-256`), they obtain a 27-round (resp. 22-round) related-tweakey rectangle attack with time complexity 2^{331} (resp. $2^{251.03}$), data of 2^{112} (resp. $2^{118.92}$) chosen plaintexts and 2^{144} (resp. 2^{120}) memory. The rectangle attacks use actual differential trails with their exact probability. The results indicate that the actual probability of the best differential trails gets lower than estimated by the number of active Sboxes as the number of rounds increases.

In [39], the authors analyze different `SKINNY` variants with regard to zero-correlation and related-tweakey impossible differential attacks. For `SKINNY-128-256`, they obtain a related-tweakey impossible differential attack on 23 rounds with time complexity of $2^{243.41}$, data of $2^{124.41}$ chosen plaintexts and $2^{155.41}$ memory. They utilize a 15-round related-tweakey impossible differential.

In [46], the authors apply impossible differential cryptanalysis on `SKINNY` in the single-key model. They utilize the 11-round impossible differential described in the design document. They obtain a key-recovery attack of 20 rounds `SKINNY-128-256` with time complexity $2^{245.72}$, data of $2^{92.1}$ chosen plaintexts and memory $2^{147.1}$. They further attack 22 rounds of `SKINNY-128-384` with time complexity $2^{373.48}$, data $2^{92.22}$ and memory $2^{147.22}$.

In [42], the authors used constrained programming for applying the Demirci-Selcuk meet-in-the-middle attack. The authors find an 10.5-round distinguisher and a 22-round key-recovery attack on `SKINNY-128-384` with time complexity $2^{382.46}$, data complexity of 2^{96} chosen plaintexts and memory complexity of $2^{330.99}$.

In [13], the authors introduce the Boomerang Connectivity Table (BCT) that quantify the boomerang switching effect in Sboxes wit regard to the boomerang attack. They apply their method to `SKINNY` and show that the probabilities of the attacks presented in [31] are not precise.

In [1], the authors proposed a method to model the actual DDT of large S-boxes in order to compute exact probabiliites of differential trails. Applied to SKINNY-128, the authors showed that the probability of any 14-round (single-key) differential trail is upper bounded by 2^{-128}, while the designers proved a lower bound of 61 active S-boxes (ensuring only a probability upper bounded by 2^{-122}).

In [29], the authors present algorithms for finding subspace trails. They find 5-round subspace trails for both SKINNY-64 and SKINNY-128.

In [2], the authors conduct an exhaustive search over all possible word permutations to be used as a replacement for the ShiftRows permutation and derived the minimum number of active S-boxes with regard to differential cryptanalysis using Matsuis branch-and-bound algorithm. Their results show that the ShiftRows permutation used in SKINNY is actually among the best permutations. By using Matsuis algorithm, they computed the bounds for up to 40 rounds in the single-key setting, while the designers only gave bounds for up to 22 rounds. Table 7 is updated with their improved bounds starting from 23 rounds.

As a summary, Table 8 shows the maximum number of rounds that can be attacked so far. Both of the underlying primitives SKINNY-128-256 and SKINNY-128-384 still offer a security margin above 50%.

Table 8: Number of rounds of SKINNY that can be attacked by the best key-recovery attacks (in the related-tweakey model) known so far.

SKINNY-128-256	SKINNY-128-384
23/48	27/56
47.9%	48.2%

6 Performance

6.1 Estimating Area and Performances

In order to discuss the rationale of our design, we first quickly describe an estimation in Gate Equivalent (GE) of the ASIC area cost of several simple bit operations (for IBM 130 nm): a NOR/NAND gate costs 1 GE, a OR/AND gate costs 1.25 GE, a XOR/XNOR gate costs 2 GE and a NOT gate costs 0.75 GE. Finally, one memory bit can be estimated to 5.5 GE (scan flip-flop). Of course, these numbers depend on the library used, but it will give us at least some rough and easy evaluation of the design choices we will make.

Besides, even though many tradeoffs exist, we distinguish between a serial implementation, a round-based implementation and a low-latency implementation. In the latter, the entire ciphering process is performed in a single clock cycle, but the area cost is then quite important as all rounds need to be directly implemented. For a round-based implementation, an entire round of the cipher is performed in a single clock cycle, thus ending with the entire ciphering process being done in r cycles and with a moderate area cost (this tradeoff is usually a good candidate for energy efficiency). Finally, in a serial implementation, one reduces the datapath and thus the area to the minimum (usually a few bits, like the Sbox bit size), but the throughput is greatly reduced. The ultimate goal of a good lightweight encryption primitive is to use lightweight components, but also to ensure that these components are compact and efficient for all these tradeoffs. This is what SIMON designers have managed to produce, but sacrificing a few security guarantees. SKINNY offers similar (sometimes even better) performances than SIMON, while providing much stronger security arguments with regard to classical differential or linear cryptanalysis.

Considering such tradeoffs, several implementations of SKINNY based on different design architectures are available and publicly accessible on the official website [6]. Notably, its structure enables conducting the operations in a bit-wise fashion [20]. More precisely, it is possible to slide the state register one bit per clock cycle and partially perform the operations. This leads to smallest area requirement (of around 800 GE) with expectedly higher latency compared to other design architectures. This can even be scaled with sliding and updating 2 or 4 bits at every clock cycle, contributing to the area-latency tradeoff.

6.2 Comparing Theoretical Performance

After some minimal security guarantee, the second design goal of SKINNY was to minimize the total number of operations. We provide in Table 9 a comparison of the total number of operations per bit for SKINNY and for other lightweight block ciphers, as well as some quality grade regarding its ASIC area in a round-based implementation.

Table 9: Total number of operations and theoretical performance of SKINNY and various lightweight block ciphers. N denotes a NOR gate, A denotes a AND gate, X denotes a XOR gate.

Cipher	nb. of rds	gate cost (per bit per round)			nb. of op.	nb. of op.	round-based
		int. cipher	key sch.	total	w/o key sch.	w/ key sch.	impl. area
SKINNY -64-128	36	1 N 2.25 X	0.625 X	1 N 2.875 X	3.25×36 $= 117$	3.875×36 $=$ **139.5**	$1 + 2.67 \times 2.875$ $=$ **8.68**
SIMON -64/128	44	0.5 A 1.5 X	1.5 X	0.5 A 3.0 X	2×44 $= 88$	3.5×44 $=$ **154**	$0.67 + 2.67 \times 3$ $=$ **8.68**
PRESENT -128	31	1 A 3.75 X	0.125 A 0.344 X	1.125 A 4.094 X	4.75×31 $= 147.2$	5.22×31 $=$ **161.8**	$1.5 + 2.67 \times 4.094$ $=$ **12.43**
PICCOLO -128	31	1 N 4.25 X		1 N 4.25 X	5.25×31 $= 162.75$	5.25×31 $=$ **162.75**	$1 + 2.67 \times 4.25$ $=$ **12.35**
KATAN -64-80	254	0.047 N 0.094 X	3 X	0.047 N 3.094 X	0.141×254 $= 35.81$	3.141×254 $=$ **797.8**	$0.19 + 2.67 \times 3.094$ $=$ **8.45**
SKINNY -128-128	40	1 N 2.25 X		1 N 2.25 X	3.25×40 $= 130$	3.25×40 $=$ **130**	$1 + 2.67 \times 2.25$ $=$ **7.01**
SIMON -128/128	68	0.5 A 1.5 X	1 X	0.5 A 2.5 X	2×68 $= 136$	3×68 $=$ **204**	$0.67 + 2.67 \times 2.5$ $=$ **7.34**
NOEKEON -128	16	0.5 (A + N) 5.25 X	0.5 (A + N) 5.25 X	1 (A + N) 10.5 X	6.25×16 $= 100$	12.5×16 $=$ **200**	$2.33 + 2.67 \times 10.5$ $=$ **30.36**
AES -128	10	4.25 A 16 X	1.06 A 3.5 X	5.31 A 19.5 X	20.25×10 $= 202.5$	24.81×10 $=$ **248.1**	$7.06 + 2.67 \times 19.5$ $=$ **59.12**
SKINNY -128-256	48	1 N 2.25 X	0.56 X	1 N 2.81 X	3.25×48 $= 156$	3.81×48 $=$ **183**	$1 + 2.67 \times 2.81$ $=$ **8.5**
SIMON -128/256	72	0.5 A 1.5 X	1.5 X	0.5 A 3.0 X	2×72 $= 144$	3.5×72 $=$ **252**	$0.67 + 2.67 \times 3$ $=$ **8.68**
AES -256	14	4.25 A 16 X	2.12 A 7 X	6.37 A 23 X	20.25×14 $= 283.5$	29.37×14 $=$ **411.2**	$8.47 + 2.67 \times 23$ $=$ **69.88**

One can see from the Table 9 that SIMON and SKINNY compare very favorably to other candidates, both in terms of number of operations and theoretical area grade for round-based implementations. This seems to confirm that when it comes to lightweight block ciphers, SIMON is probably the strongest competitor as of today. Besides, SKINNY

has the best theoretical profile among all the candidates presented here, even better than SIMON for area. For speed efficiency, SKINNY outperforms SIMON when the key schedule is taken in account. This scenario is arguably the most important in practice: as remarked in [9], it is likely that lightweight devices will cipher very small messages and thus the back-end servers communicating with millions of devices will probably have to recompute the key schedule for every small message received.

In addition to its smaller key size, we note that KATAN-64-80 [12] theoretical area grade is slightly biased here as one round of this cipher is extremely light and such a round-based implementation would actually look more like a serial implementation and will have a very low throughput (KATAN-64-80 has 254 rounds in total).

While Table 9 is only a rough indication of the efficiency of the various designs, we observe that the ratio between the SIMON and SKINNY best software implementations, or the ratio between the smallest SIMON and SKINNY round-based hardware implementations actually match the results from the table.

6.3 Hardware Implementations of SKINNY-AEAD and SKINNY-Hash Members

We also provide performance results and area footprints of SKINNY-AEAD and SKINNY-Hash when implemented on a hardware platform. To this end, in addition to the tk3 and tk2 constructions of SKINNY-Hash, we realized two sets of implementations for each SKINNY-AEAD variant: one as encryption-only and the other one supporting both encryption and decryption functionalities. We further considered two different instances of the underlying SKINNY module: a round-based implementation performing every cipher round in a clock cycle, and a byte-serial implementation mainly processing a single byte per clock cycle. The corresponding results are depicted in Table 10 and Table 12, respectively, where the area footprint (in GE), maximum clock frequency, and maximum throughput for two standard cell libraries IBM 130 and UMC 90 are reported. In order to achieve the maximum throughput, we simulated the SKINNY-AEAD implementations with 100 blocks of 16-byte associated data A and 100 blocks of 16-byte message M, thereby obtaining the required number of clock cycles. Note that the number of clock cycles is independent of the value of the given associated data and the message as our implementations are constant-time preventing any leakage through the timing side channel. For SKINNY-Hash we obtained the number of required clock cycles by simulating the implementation with a message of 100 blocks of 16-byte (for SKINNY-tk3-Hash) and a message of 100 blocks of 4-byte (for SKINNY-tk2-Hash).

We further realized the side-channel protected variants of all aforementioned implementations. We applied a masking countermeasure with 2 shares and made use of the concept of Domain-Oriented Masking (DOM) [17] to provide secure implementations. It is noteworthy that due to the special construction of the SKINNY Sbox, its SCA-protected version by DOM does not require any fresh randomness. Therefore, in all variants of SKINNY-AEAD, it is sufficient to present the entire inputs and output (including the associated data, message, key, tag and output) by two uniformly-masked additive shares. It also holds for SKINNY-Hash, while SKINNY-tk3-Hash requires an additional 384-bit of initial mask used to initialize the state S_{384} in a shared way with 2 shares. Trivially, SKINNY-tk3-Hash also needs such an initial mask (of 256 bits). The performance results and the area footprint of all implementations are given in Table 11 and Table 13.

7 Intellectual Property

SKINNY is not patented and is free for use in any application. We note that since SKINNY-AEAD 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. 7,949,129; United States Patent No.8,321,675) apply to our proposal.

Table 10: Unprotected, round-based ASIC implementations of our SKINNY-AEAD and SKINNY-Hash members using IBM 130 and UMC 90 standard cell libraries.

	IBM 130			UMC 90		
	Area	**Freq.**	**Throughput**	**Area**	**Freq.**	**Throughput**
	GE	MHz	MBit/s	GE	MHz	MBit/s
SKINNY-AEAD-M1-Enc	7627	184	406	7808	269	594
SKINNY-AEAD-M2-Enc	7595	181	400	7808	287	633
SKINNY-AEAD-M3-Enc	7100	189	418	7291	267	589
SKINNY-AEAD-M4-Enc	7069	164	363	7266	233	514
SKINNY-AEAD-M5-Enc	6512	171	428	6636	298	743
SKINNY-AEAD-M6-Enc	5953	208	519	6099	288	720
SKINNY-AEAD-M1-Enc-Dec	10370	158	349	10239	227	501
SKINNY-AEAD-M2-Enc-Dec	10318	166	367	10210	240	530
SKINNY-AEAD-M3-Enc-Dec	9817	167	368	9671	244	539
SKINNY-AEAD-M4-Enc-Dec	9772	164	362	9564	202	447
SKINNY-AEAD-M5-Enc-Dec	8844	166	416	8894	268	670
SKINNY-AEAD-M6-Enc-Dec	8290	166	414	8332	222	555
SKINNY-tk3-Hash	8622	212	154	8894	285	207
SKINNY-tk2-Hash	5730	211	66	6019	304	95

Table 11: SCA-protected (2 shares), round-based ASIC implementations of our SKINNY-AEAD and SKINNY-Hash members using IBM 130 and UMC 90 standard cell libraries.

	IBM 130			UMC 90		
	Area	**Freq.**	**Throughput**	**Area**	**Freq.**	**Throughput**
	GE	MHz	MBit/s	GE	MHz	MBit/s
SKINNY-AEAD-M1-Enc	13812	427	192	14481	885	398
SKINNY-AEAD-M2-Enc	13787	427	192	14445	885	398
SKINNY-AEAD-M3-Enc	12716	427	192	13422	885	398
SKINNY-AEAD-M4-Enc	12692	427	192	13383	885	398
SKINNY-AEAD-M5-Enc	12653	435	228	13263	885	464
SKINNY-AEAD-M6-Enc	11555	422	221	12203	885	464
SKINNY-AEAD-M1-Enc-Dec	18817	446	201	20534	909	409
SKINNY-AEAD-M2-Enc-Dec	18768	444	200	20460	917	413
SKINNY-AEAD-M3-Enc-Dec	17723	446	201	19474	909	409
SKINNY-AEAD-M4-Enc-Dec	17675	444	200	19400	917	413
SKINNY-AEAD-M5-Enc-Dec	17285	435	228	18888	820	430
SKINNY-AEAD-M6-Enc-Dec	16180	448	235	17828	820	430
SKINNY-tk3-Hash	18388	429	64	19178	787	118
SKINNY-tk2-Hash	12404	405	26	13041	794	52

Table 12: Unprotected, byte-serial ASIC implementations of our SKINNY-AEAD and SKINNY-Hash members using IBM 130 and UMC 90 standard cell libraries.

	IBM 130			UMC 90		
	Area	**Freq.**	**Throughput**	**Area**	**Freq.**	**Throughput**
	GE	MHz	MBit/s	GE	MHz	MBit/s
SKINNY-AEAD-M1-Enc	7270	532	57	7253	1124	121
SKINNY-AEAD-M2-Enc	7238	532	57	7237	1124	121
SKINNY-AEAD-M3-Enc	6723	418	45	6709	654	71
SKINNY-AEAD-M4-Enc	6690	418	45	6707	654	71
SKINNY-AEAD-M5-Enc	6157	439	55	6199	1429	180
SKINNY-AEAD-M6-Enc	5601	478	60	5669	1429	180
SKINNY-AEAD-M1-Enc-Dec	9554	291	31	9038	327	35
SKINNY-AEAD-M2-Enc-Dec	9485	228	25	8939	340	37
SKINNY-AEAD-M3-Enc-Dec	9002	284	31	8516	392	42
SKINNY-AEAD-M4-Enc-Dec	8947	258	28	8428	524	57
SKINNY-AEAD-M5-Enc-Dec	8002	264	33	7702	412	52
SKINNY-AEAD-M6-Enc-Dec	7456	267	34	7179	422	53
SKINNY-tk3-Hash	8406	485	17	8405	741	26
SKINNY-tk2-Hash	5554	402	6	5484	538	8

Table 13: SCA-protected (2 shares), byte-serial ASIC implementations of our SKINNY-AEAD and SKINNY-Hash members using IBM 130 and UMC 90 standard cell libraries.

	IBM 130			UMC 90		
	Area	**Freq.**	**Throughput**	**Area**	**Freq.**	**Throughput**
	GE	MHz	MBit/s	GE	MHz	MBit/s
SKINNY-AEAD-M1-Enc	13289	287	8	13939	424	11
SKINNY-AEAD-M2-Enc	13265	287	8	13913	424	11
SKINNY-AEAD-M3-Enc	12195	287	8	12881	424	11
SKINNY-AEAD-M4-Enc	12171	287	8	12852	424	11
SKINNY-AEAD-M5-Enc	12143	274	9	12756	448	14
SKINNY-AEAD-M6-Enc	11048	274	9	11694	448	14
SKINNY-AEAD-M1-Enc-Dec	17115	342	9	18452	478	13
SKINNY-AEAD-M2-Enc-Dec	17065	342	9	18369	478	13
SKINNY-AEAD-M3-Enc-Dec	16019	342	9	17394	478	13
SKINNY-AEAD-M4-Enc-Dec	15968	342	9	17309	478	13
SKINNY-AEAD-M5-Enc-Dec	15528	248	8	16719	610	19
SKINNY-AEAD-M6-Enc-Dec	14432	248	8	15660	610	19
SKINNY-tk3-Hash	17698	265	2	18793	505	4
SKINNY-tk2-Hash	11827	292	1	12514	348	1

References

1. Abdelkhalek, A., Sasaki, Y., Todo, Y., Tolba, M., Youssef, A.M.: MILP modeling for (large) s-boxes to optimize probability of differential characteristics. IACR Trans. Symmetric Cryptol. **2017**(4) (2017) 99–129
2. Alfarano, G.N., Beierle, C., Isobe, T., Kölbl, S., Leander, G.: ShiftRows alternatives for AES-like ciphers and optimal cell permutations for Midori and Skinny. IACR Trans. Symmetric Cryptol. **2018**(2) (2018) 20–47
3. Ankele, R., Banik, S., Chakraborti, A., List, E., Mendel, F., Sim, S.M., Wang, G.: Related-key impossible-differential attack on reduced-round Skinny. In Gollmann, D., Miyaji, A., Kikuchi, H., eds.: Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings. Volume 10355 of Lecture Notes in Computer Science., Springer (2017) 208–228
4. Ankele, R., Kölbl, S.: Mind the gap - A closer look at the security of block ciphers against differential cryptanalysis. [14] 163–190
5. Beierle, C., Canteaut, A., Leander, G., Rotella, Y.: Proving resistance against invariant attacks: How to choose the round constants. In Katz, J., Shacham, H., eds.: Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II. Volume 10402 of Lecture Notes in Computer Science., Springer (2017) 647–678
6. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: SKINNY family of block ciphers, website https://sites.google.com/site/skinnycipher/home.
7. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In Robshaw, M., Katz, J., eds.: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. Volume 9815 of Lecture Notes in Computer Science., Springer (2016) 123–153
8. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. IACR Cryptology ePrint Archive **2016** (2016) 660
9. Benadjila, R., Guo, J., Lomné, V., Peyrin, T.: Implementing lightweight block ciphers on x86 architectures. In Lange, T., Lauter, K.E., Lisonek, P., eds.: Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers. Volume 8282 of Lecture Notes in Computer Science., Springer (2013) 324–351
10. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT hash workshop. (2007)
11. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Cryptographic sponge functions. http://sponge.noekeon.org/ (2011)
12. Cannière, C.D., Dunkelman, O., Knezevic, M.: KATAN and KTANTAN - A family of small and efficient hardware-oriented block ciphers. In Clavier, C., Gaj, K., eds.: Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings. Volume 5747 of Lecture Notes in Computer Science., Springer (2009) 272–288
13. Cid, C., Huang, T., Peyrin, T., Sasaki, Y., Song, L.: Boomerang connectivity table: A new cryptanalysis tool. In Nielsen, J.B., Rijmen, V., eds.: Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II. Volume 10821 of Lecture Notes in Computer Science., Springer (2018) 683–714
14. Cid, C., Jr., M.J.J., eds.: Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers. Volume 11349 of Lecture Notes in Computer Science., Springer (2019)
15. Dworkin, M.J.: SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Inf. Process. Stds.(NIST FIPS)-202 (2015)
16. Eskandari, Z., Kidmose, A.B., Kölbl, S., Tiessen, T.: Finding integral distinguishers with ease. [14] 115–138

17. Groß, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In Bilgin, B., Nikova, S., Rijmen, V., eds.: Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016, ACM (2016) 3
18. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED block cipher. In: Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. Volume 6917 of Lecture Notes in Computer Science., Springer (2011) 326–341
19. Hall-Andersen, M., Vejre, P.S.: Generating graphs packed with paths. IACR Trans. Symmetric Cryptol. **2018**(3) (2018) 265–289
20. Jean, J., Moradi, A., Peyrin, T., Sasdrich, P.: Bit-sliding: A generic technique for bit-serial implementations of spn-based primitives - applications to aes, PRESENT and SKINNY. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Volume 10529 of Lecture Notes in Computer Science., Springer (2017) 687–707
21. Jean, J., Nikolic, I., Peyrin, T.: Tweaks and keys for block ciphers: The TWEAKEY framework. In Sarkar, P., Iwata, T., eds.: Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II. Volume 8874 of Lecture Notes in Computer Science., Springer (2014) 274–288
22. Jean, J., Nikolić, I., Peyrin, T., Seurin, Y.: Deoxys v1.41 (2016) Submission to CAESAR, available via https://competitions.cr.yp.to/round3/deoxysv141.pdf.
23. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2^n work. In Cramer, R., ed.: Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. Volume 3494 of Lecture Notes in Computer Science., Springer (2005) 474–490
24. Khoo, K., Peyrin, T., Poschmann, A.Y., Yap, H.: FOAM: searching for hardware-optimal SPN structures and components with a fair comparison. In Batina, L., Robshaw, M., eds.: Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings. Volume 8731 of Lecture Notes in Computer Science., Springer (2014) 433–450
25. Kölbl, S., Leander, G., Tiessen, T.: Observations on the SIMON block cipher family. In Gennaro, R., Robshaw, M., eds.: Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I. Volume 9215 of Lecture Notes in Computer Science., Springer (2015) 161–185
26. Kranz, T., Leander, G., Wiemer, F.: Linear cryptanalysis: Key schedules and tweakable block ciphers. IACR Trans. Symmetric Cryptol. **2017**(1) (2017) 474–505
27. Krovetz, T., Rogaway, P.: The software performance of authenticated-encryption modes. In Joux, A., ed.: Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers. Volume 6733 of Lecture Notes in Computer Science., Springer (2011) 306–327
28. Krovetz, T., Rogaway, P.: OCB (v1.1) (2016) Submission to CAESAR, available via https://competitions.cr.yp.to/round3/ocbv11.pdf.
29. Leander, G., Tezcan, C., Wiemer, F.: Searching for subspace trails and truncated differentials. IACR Trans. Symmetric Cryptol. **2018**(1) (2018) 74–100
30. Liskov, M., Rivest, R.L., Wagner, D.A.: Tweakable block ciphers. In Yung, M., ed.: Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings. Volume 2442 of Lecture Notes in Computer Science., Springer (2002) 31–46
31. Liu, G., Ghosh, M., Song, L.: Security analysis of SKINNY under related-tweakey settings (long paper). IACR Trans. Symmetric Cryptol. **2017**(3) (2017) 37–72
32. McGrew, D., Viega, J.: The Galois/Counter mode of operation (GCM). Submission to NIST. http://csrc. nist. gov/CryptoToolk it/modes/proposedmodes/gcm/gcm-spec. pdf (2004)
33. 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.: Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications

of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings. Volume 6632 of Lecture Notes in Computer Science., Springer (2011) 69–88
34. 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.: Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers. Volume 7537 of Lecture Notes in Computer Science., Springer (2011) 57–76
35. Naito, Y., Ohta, K.: Improved indifferentiable security analysis of PHOTON. In Abdalla, M., Prisco, R.D., eds.: Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings. Volume 8642 of Lecture Notes in Computer Science., Springer (2014) 340–357
36. Posteuca, R., Negara, G.: New related-key attacks and properties of SKINNY-64-128 cipher. Proceedings of the Romanian Academy, Series A **18, Special Issue 2017** (2017) 333–350
37. Rogaway, P.: Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Lee, P.J., ed.: Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings. Volume 3329 of Lecture Notes in Computer Science., Springer (2004) 16–31
38. Rogaway, P., Bellare, M., Black, J., Krovetz, T.: OCB: a block-cipher mode of operation for efficient authenticated encryption. In Reiter, M.K., Samarati, P., eds.: CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001., ACM (2001) 196–205
39. Sadeghi, S., Mohammadi, T., Bagheri, N.: Cryptanalysis of reduced round SKINNY block cipher. IACR Trans. Symmetric Cryptol. **2018**(3) (Sep. 2018) 124–162
40. Sasaki, Y., Todo, Y.: New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In Coron, J., Nielsen, J.B., eds.: Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III. Volume 10212 of Lecture Notes in Computer Science. (2017) 185–215
41. Schroeppel, R.: The Hasty Pudding Cipher. NIST AES proposal (1998)
42. Shi, D., Sun, S., Derbez, P., Todo, Y., Sun, B., Hu, L.: Programming the demirci-selçuk meet-in-the-middle attack with constraints. In Peyrin, T., Galbraith, S.D., eds.: Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II. Volume 11273 of Lecture Notes in Computer Science., Springer (2018) 3–34
43. Shibutani, K., Isobe, T., Hiwatari, H., Mitsuda, A., Akishita, T., Shirai, T.: Piccolo: An ultra-lightweight blockcipher. In: Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings. Volume 6917 of Lecture Notes in Computer Science., Springer (2011) 342–357
44. Sun, S., Gerault, D., Lafourcade, P., Yang, Q., Todo, Y., Qiao, K., Hu, L.: Analysis of AES, SKINNY, and others with constraint programming. IACR Trans. Symmetric Cryptol. **2017**(1) (2017) 281–306
45. Sun, S., Hu, L., Song, L., Xie, Y., Wang, P.: Automatic security evaluation of block ciphers with s-bp structures against related-key differential attacks. In Lin, D., Xu, S., Yung, M., eds.: Information Security and Cryptology - 9th International Conference, Inscrypt 2013, Guangzhou, China, November 27-30, 2013, Revised Selected Papers. Volume 8567 of Lecture Notes in Computer Science., Springer (2013) 39–51
46. Tolba, M., Abdelkhalek, A., Youssef, A.M.: Impossible differential cryptanalysis of reduced-round SKINNY. In Joye, M., Nitaj, A., eds.: Progress in Cryptology - AFRICACRYPT 2017 - 9th International Conference on Cryptology in Africa, Dakar, Senegal, May 24-26, 2017, Proceedings. Volume 10239 of Lecture Notes in Computer Science. (2017) 117–134
47. Vaudenay, S., Vizár, D.: Under pressure: Security of caesar candidates beyond their guarantees. IACR Cryptology ePrint Archive **2017** (2017) 1147
48. Yang, D., Qi, W., Chen, H.: Impossible differential attacks on the SKINNY family of block ciphers. IET Information Security **11**(6) (2017) 377–385
49. Zhang, P., Zhang, W.: Differential cryptanalysis on block cipher skinny with MILP program. Security and Communication Networks **2018** (2018) 3780407:1–3780407:11

50. Zhang, W., Rijmen, V.: Division cryptanalysis of block ciphers with a binary diffusion layer. IET Information Security (August 2018)
51. Zheng, Y., Wu, W.: Biclique attack of block cipher SKINNY. In Chen, K., Lin, D., Yung, M., eds.: Information Security and Cryptology - 12th International Conference, Inscrypt 2016, Beijing, China, November 4-6, 2016, Revised Selected Papers. Volume 10143 of Lecture Notes in Computer Science., Springer (2016) 3–17

A The 8-bit Sbox for SKINNY

```
/* SKINNY Sbox */
uint8_t S8[256] = {
 0x65,0x4c,0x6a,0x42,0x4b,0x63,0x43,0x6b,0x55,0x75,0x5a,0x7a,0x53,0x73,0x5b,0x7b,
 0x35,0x8c,0x3a,0x81,0x89,0x33,0x80,0x3b,0x95,0x25,0x98,0x2a,0x90,0x23,0x99,0x2b,
 0xe5,0xcc,0xe8,0xc1,0xc9,0xe0,0xc0,0xe9,0xd5,0xf5,0xd8,0xf8,0xd0,0xf0,0xd9,0xf9,
 0xa5,0x1c,0xa8,0x12,0x1b,0xa0,0x13,0xa9,0x05,0xb5,0x0a,0xb8,0x03,0xb0,0x0b,0xb9,
 0x32,0x88,0x3c,0x85,0x8d,0x34,0x84,0x3d,0x91,0x22,0x9c,0x2c,0x94,0x24,0x9d,0x2d,
 0x62,0x4a,0x6c,0x45,0x4d,0x64,0x44,0x6d,0x52,0x72,0x5c,0x7c,0x54,0x74,0x5d,0x7d,
 0xa1,0x1a,0xac,0x15,0x1d,0xa4,0x14,0xad,0x02,0xb1,0x0c,0xbc,0x04,0xb4,0x0d,0xbd,
 0xe1,0xc8,0xec,0xc5,0xcd,0xe4,0xc4,0xed,0xd1,0xf1,0xdc,0xfc,0xd4,0xf4,0xdd,0xfd,
 0x36,0x8e,0x38,0x82,0x8b,0x30,0x83,0x39,0x96,0x26,0x9a,0x28,0x93,0x20,0x9b,0x29,
 0x66,0x4e,0x68,0x41,0x49,0x60,0x40,0x69,0x56,0x76,0x58,0x78,0x50,0x70,0x59,0x79,
 0xa6,0x1e,0xaa,0x11,0x19,0xa3,0x10,0xab,0x06,0xb6,0x08,0xba,0x00,0xb3,0x09,0xbb,
 0xe6,0xce,0xea,0xc2,0xcb,0xe3,0xc3,0xeb,0xd6,0xf6,0xda,0xfa,0xd3,0xf3,0xdb,0xfb,
 0x31,0x8a,0x3e,0x86,0x8f,0x37,0x87,0x3f,0x92,0x21,0x9e,0x2e,0x97,0x27,0x9f,0x2f,
 0x61,0x48,0x6e,0x46,0x4f,0x67,0x47,0x6f,0x51,0x71,0x5e,0x7e,0x57,0x77,0x5f,0x7f,
 0xa2,0x18,0xae,0x16,0x1f,0xa7,0x17,0xaf,0x01,0xb2,0x0e,0xbe,0x07,0xb7,0x0f,0xbf,
 0xe2,0xca,0xee,0xc6,0xcf,0xe7,0xc7,0xef,0xd2,0xf2,0xde,0xfe,0xd7,0xf7,0xdf,0xff
};


/* Inverse SKINNY Sbox */
uint8_t S8_inv[256] = {
  0xac,0xe8,0x68,0x3c,0x6c,0x38,0xa8,0xec,0xaa,0xae,0x3a,0x3e,0x6a,0x6e,0xea,0xee,
  0xa6,0xa3,0x33,0x36,0x66,0x63,0xe3,0xe6,0xe1,0xa4,0x61,0x34,0x31,0x64,0xa1,0xe4,
  0x8d,0xc9,0x49,0x1d,0x4d,0x19,0x89,0xcd,0x8b,0x8f,0x1b,0x1f,0x4b,0x4f,0xcb,0xcf,
  0x85,0xc0,0x40,0x15,0x45,0x10,0x80,0xc5,0x82,0x87,0x12,0x17,0x42,0x47,0xc2,0xc7,
  0x96,0x93,0x03,0x06,0x56,0x53,0xd3,0xd6,0xd1,0x94,0x51,0x04,0x01,0x54,0x91,0xd4,
  0x9c,0xd8,0x58,0x0c,0x5c,0x08,0x98,0xdc,0x9a,0x9e,0x0a,0x0e,0x5a,0x5e,0xda,0xde,
  0x95,0xd0,0x50,0x05,0x55,0x00,0x90,0xd5,0x92,0x97,0x02,0x07,0x52,0x57,0xd2,0xd7,
  0x9d,0xd9,0x59,0x0d,0x5d,0x09,0x99,0xdd,0x9b,0x9f,0x0b,0x0f,0x5b,0x5f,0xdb,0xdf,
  0x16,0x13,0x83,0x86,0x46,0x43,0xc3,0xc6,0x41,0x14,0xc1,0x84,0x11,0x44,0x81,0xc4,
  0x1c,0x48,0xc8,0x8c,0x4c,0x18,0x88,0xcc,0x1a,0x1e,0x8a,0x8e,0x4a,0x4e,0xca,0xce,
  0x35,0x60,0xe0,0xa5,0x65,0x30,0xa0,0xe5,0x32,0x37,0xa2,0xa7,0x62,0x67,0xe2,0xe7,
  0x3d,0x69,0xe9,0xad,0x6d,0x39,0xa9,0xed,0x3b,0x3f,0xab,0xaf,0x6b,0x6f,0xeb,0xef,
  0x26,0x23,0xb3,0xb6,0x76,0x73,0xf3,0xf6,0x71,0x24,0xf1,0xb4,0x21,0x74,0xb1,0xf4,
  0x2c,0x78,0xf8,0xbc,0x7c,0x28,0xb8,0xfc,0x2a,0x2e,0xba,0xbe,0x7a,0x7e,0xfa,0xfe,
  0x25,0x70,0xf0,0xb5,0x75,0x20,0xb0,0xf5,0x22,0x27,0xb2,0xb7,0x72,0x77,0xf2,0xf7,
  0x2d,0x79,0xf9,0xbd,0x7d,0x29,0xb9,0xfd,0x2b,0x2f,0xbb,0xbf,0x7b,0x7f,0xfb,0xff
};
```

B Test Vectors for SKINNY-128-256 and SKINNY-128-384

```
/* Skinny-128-256 */
Tweakey:    009cec81605d4ac1d2ae9e3085d7a1f3
            1ac123ebfc00fddcf01046ceeddfcab3
Plaintext:  3a0c47767a26a68dd382a695e7022e25
Ciphertext: b731d98a4bde147a7ed4a6f16b9b587f

/* Skinny-128-384 */
Tweakey:    df889548cfc7ea52d296339301797449
            ab588a34a47f1ab2dfe9c8293fbea9a5
            ab1afac2611012cd8cef952618c3ebe8
Plaintext:  a3994b66ad85a3459f44e92b08f550cb
Ciphertext: 94ecf589e2017c601b38c6346a10dcfa
```