SIV-TEM-PHOTON Authenticated Encryption and Hash Family

Designers/Submitters:

Zhenzhen Bao - Nanyang Technological University, Singapore
Jian Guo - Nanyang Technological University, Singapore
Tetsu Iwata - Nagoya University, Japan
Ling Song - Nanyang Technological University, Singapore and Institute of Information Engineering, CAS, China

zzbao@ntu.edu.sg, guojian@ntu.edu.sg, tetsu.iwata@nagoya-u.jp, songling@ntu.edu.sg

February 25, 2019

Chapter 1

Specification

In this document we propose the SIV-TEM-PHOTON family of AEAD and hash function, which utilizes components of PHOTON as the underlying building block. On top of the round functions of PHOTON, a tweakable block cipher named TEM-PHOTON with 128-bit key and 128-bit tweak is proposed. Combined with the SIV mode, SIV-TEM-PHOTON-AEAD authenticated encryption is built. By setting the key and tweak of the TEM-PHOTON to a constant, a permutation is obtained, based on which a Sponge hash function named SIV-TEM-PHOTON-hash is proposed. Both the AEAD and hash enjoys the long-standing security, as well as the lightweightness of PHOTON.

1.1 Notations and Preliminaries

1.1.1 Notations

Let $\{0,1\}^*$ be the set of all finite bit strings, including the empty string ε. For a bit string $X \in \{0,1\}^*$, $|X|$ is its length in bits, and we have $|\varepsilon| = 0$. For a bit string $X \in \{0,1\}^*$ and an integer $n \geq 1$, we define a parsing operation. For $X \neq \varepsilon$, it is defined as $(X[1], \ldots, X[x]) \xleftarrow{n} X$, where $|X[i]| = n$ for $1 \leq i \leq x-1$, $1 \leq |X[x]| \leq n$, and $X[1] \| \cdots \| X[x] = X$. Here $X \| Y$ is the concatenation of two bit strings X and Y. The number of blocks, x, is the block length of X. For $X = \varepsilon$, $X[1] \xleftarrow{n} X$, where $X[1] = \varepsilon$. Note that $x = 1$ and the block length of $X = \varepsilon$ is 1. For a bit string $X \in \{0,1\}^*$ and two positive integers n_1, n_2, we define a similar parsing operation. If $|X| > n_1$, it is defined as $(X[1], \ldots, X[x]) \xleftarrow{n_1, n_2} X$, where $|X[1]| = n_1$, $|X[2]| = \cdots = |X[x-1]| = n_2$, $1 \leq |X[x]| \leq n_2$, and $X[1] \| \cdots \| X[x] = X$. If $|X| \leq n_1$, including $X = \varepsilon$, $(X[1], \ldots, X[x]) \xleftarrow{n_1, n_2} X$ is equivalent to $X[1] \leftarrow X$ and $x = 1$. For a bit string $X \in \{0,1\}^*$ and an integer $\ell \leq |X|$, $\mathrm{msb}_\ell(X)$ denotes the first ℓ bits of X and $\mathrm{lsb}_\ell(X)$ denotes the last ℓ bits of X.

For $X \in \{0,1\}^*$ with $|X| \leq \ell$, we define a padding function as $\mathrm{pad}_\ell(X) = X$ if $|X| = \ell$, and $\mathrm{pad}_\ell(X) = X \| 10^{\ell - 1 - (|X| \bmod \ell)}$ if $0 \leq |X| < \ell$.

1.1.2 Synthetic Initialization Vector Scheme (SIV-scheme)

SIV scheme [13] combines an encryption scheme $\mathcal{E}$ and a pseudorandom function (PRF) $\mathcal{F}$ to obtain an AEAD scheme. We modify the original scheme in two ways.

- We modify the PRF so that it explicitly takes a nonce N as a part of the input.
- The encryption scheme and the PRF share the same key, and we maintain their independence with domain separation.

Fix the key length k and a block length n. The encryption scheme $\mathcal{E}$ takes a key $K \in \{0,1\}^k$, initial value (IV) $IV \in \{0,1\}^n$, and a plaintext $M \in \{0,1\}^*$ as input, and returns a ciphertext $C \in \{0,1\}^{|M|}$, and we write $C = \mathcal{E}_K^{IV}(M)$. The corresponding decryption scheme $\mathcal{D}$ takes (K, IV, C) and returns M, and we write $M = \mathcal{D}_K^{IV}(C)$. We require, for any K and IV, $M = \mathcal{D}_K^{IV}(\mathcal{E}_K^{IV}(M))$.

The PRF $\mathcal{F}$ takes a key $K \in \{0,1\}^k$, a nonce $N \in \{0,1\}^*$, associated data (AD) $A \in \{0,1\}^*$, and a plaintext $M \in \{0,1\}^*$ as input, and returns a fixed length output $T \in \{0,1\}^n$, and we write $T = \mathcal{F}_K(N, A, M)$.

With these components, the encryption algorithm of SIV scheme $\mathsf{SIV.Enc}$ takes a key K, a nonce N, AD A, and a plaintext M as input, and returns a pair of ciphertext and tag (C, T). We write $(C, T) = \mathsf{SIV.Enc}_K(N, A, M)$. The decryption algorithm of SIV scheme $\mathsf{SIV.Dec}$ takes (K, N, A, C, T) as input, and returns the corresponding plaintext M or the symbol $\perp$ indicating rejection. We write $M = \mathsf{SIV.Dec}_K(N, A, C, T)$ or $\perp = \mathsf{SIV.Dec}_K(N, A, C, T)$. They are define in Fig. 1.1.

```
Algorithm $\mathsf{SIV.Enc}_K(N, A, M)$
    $T \leftarrow \mathcal{F}_K(N, A, M)$
    $C \leftarrow \mathcal{E}_K^T(M)$
    return $(C, T)$
```

```
Algorithm $\mathsf{SIV.Dec}_K(N, A, C, T)$
    $M \leftarrow \mathcal{D}_K^T(C)$
    $T^* \leftarrow \mathcal{F}_K(N, A, M)$
    if $T^* = T$ then return $M$
    else return $\perp$
```

Figure 1.1: The encryption and decryption algorithms of SIV scheme.

Let $E : \{0,1\}^k \times (\mathcal{I} \times \{0,1\}^t) \times \{0,1\}^n \to \{0,1\}^n$ be the underlying TBC, where k is the key length, $\mathcal{I}$ is the domain separation space, t is the tweak length, and n is the block length. We instantiate $\mathcal{E}$ and $\mathcal{D}$ as in Fig. 1.2. This is an OFB mode of E, where the tweak is fixed to 0^t. See the overall illustration in Fig. 1.4.

```
Algorithm $\mathcal{E}_K^{IV}(M)$
    $(M[1], \dots, M[m]) \xleftarrow{n} M$
    $S \leftarrow IV$
    for $i = 1$ to $m - 1$
        $S \leftarrow E_K^{7,0^t}(S)$
        $C[i] \leftarrow S \oplus M[i]$
    $S \leftarrow E_K^{7,0^t}(S)$
    $C[m] \leftarrow \mathrm{msb}_{|M[m]|}(S) \oplus M[m]$
    $C \leftarrow (C[1], \dots, C[m])$
    return $C$
```

```
Algorithm $\mathcal{D}_K^{IV}(C)$
    $(C[1], \dots, C[m]) \xleftarrow{n} C$
    $S \leftarrow IV$
    for $i = 1$ to $m - 1$
        $S \leftarrow E_K^{7,0^t}(S)$
        $M[i] \leftarrow S \oplus C[i]$
    $S \leftarrow E_K^{7,0^t}(S)$
    $M[m] \leftarrow \mathrm{msb}_{|C[m]|}(S) \oplus C[m]$
    $M \leftarrow (M[1], \dots, M[m])$
    return $M$
```

Figure 1.2: The definitions of $\mathcal{E}$ and $\mathcal{D}$.

The definition of $\mathcal{F}$ is presented in Fig. 1.3. This is a variant of CBC-MAC, where we process AD blocks by using the tweak input of the underlying TBC.

1.1.3 Tweakable Even-Mansour (TEM)

The Even-Mansor construction is an easy way to construct a block cipher from a fixed open permutation [5]. The simplest construction is defined as $E_{K_1,K_2}(x) = K_2 \oplus P(x \oplus K_1)$, given the open permutation P, and keys K_1, K_2. This is the case when there is only one iteration, and the security strength of the variant of r iterations ($K_{r+1} \oplus P_r(K_{r-1} \oplus P_{r-1}(\cdots(P_1(x \oplus K_1))))$) is proven to be $rn/(r+1)$ bits [3]. The extension to tweakable block ciphers with 3 iterations and single-key $E(K, T, x) = K \oplus T \oplus P_3(K \oplus T \oplus P_2(K \oplus T \oplus P_1(K \oplus T \oplus x)))$ is proven to offer birthday security, i.e., $n/2$ bits, by Cogliati *et al.* [4]. In our design TEM-PHOTON, the needs are different: a TBC with block length 256 bits and key size 128 bits are needed, we extend the tweakable Even-Mansour construction to 4 iterations (to be conservative) with the key constructed by $K||K$ (the concatenation of the same 128-bit key twice) and the tweak by $0^{128}||T$, as depicted in Fig. 1.5. We will show, by the security analysis, the probability of internal differentials (trying to exploit the fact the two halves of the key are the same) is too small to be utilized in any attack.

1.1.4 PHOTON Permutation

The PHOTON permutation [7] is a family of fixed-key AES-like functions used in the PHOTON hash function. It has five instances denoted P_t, with state sizes being $t = 100, 144, 196, 256$ and 288 bits respectively. In SIV-TEM-PHOTON, P_{256} is used and will be referred to as the PHOTON permutation for simplicity in this document.

```
Algorithm $\mathcal{F}_K(N, A, M)$
$S \leftarrow 0^n$
$(A[1], \ldots, A[a]) \overset{n+t}{\leftarrow} A$
if $|A[a]| < n + t$ then $d \leftarrow 1$ else $d \leftarrow 2$
$A[a] \leftarrow \mathrm{pad}_{n+t}(A[a])$
for $i = 1$ to $a$ do
    $S \leftarrow S \oplus \mathrm{msb}_n(A[i])$
    $S \leftarrow E_K^{0,\mathrm{lsb}_t(A[i])}(S)$
$(M[1], \ldots, M[m]) \overset{n+t}{\leftarrow} M$
for $i = 1$ to $m - 1$ do
    $S \leftarrow S \oplus \mathrm{msb}_n(M[i])$
    $S \leftarrow E_K^{d,\mathrm{lsb}_t(M[i])}(S)$
if $|M[m]| < n$ then
    $S \leftarrow S \oplus \mathrm{pad}_n(M[m])$
    $T \leftarrow E_K^{3,N}(S)$
if $|M[m]| = n$ then
    $S \leftarrow S \oplus M[m]$
    $T \leftarrow E_K^{4,N}(S)$
if $n < |M[m]| < n + t$ then
    $M[m] \leftarrow \mathrm{pad}_{n+t}(M[m])$
    $S \leftarrow S \oplus \mathrm{msb}_n(M[m])$
    $S \leftarrow E_K^{d,\mathrm{lsb}_t(M[m])}(S)$
    $T \leftarrow E_K^{5,N}(S)$
if $|M[m]| = n + t$ then
    $S \leftarrow S \oplus \mathrm{msb}_n(M[m])$
    $S \leftarrow E_K^{d,\mathrm{lsb}_t(M[m])}(S)$
    $T \leftarrow E_K^{6,N}(S)$
return $T$
```

Figure 1.3: The definitions of $\mathcal{F}$.

The internal state of P_{256} can be seen as an 8×8 array of 4-bit cells, where the cell located at row i and column j is denoted $S[i,j]$ with $0 \leq i,j < 8$. P_{256} iterates a round function N_r times and N_r is 12 in the original P_{256} while it is 20 in SIV-TEM-PHOTON. Each round function consists of four operations (see Fig. 1.6): AddConstant[r], SubCells, ShiftRows and MixColumnSerial.

- AddConstant[r] consists in adding fixed values to the first column of the internal state. Concretely, $S[i,0] \leftarrow S[i,0] \oplus IC[i] \oplus RC[r]$ for all $0 \leq i < 8$, where the internal constant $IC = [0, 1, 3, 7, 15, 14, 12, 8]$ and $RC[r]$ is a 4-bit round constant for round r. While $RC[r]$ in PHOTON is generated by a 4-bit linear feedback shift register, we use a 6-bit linear feedback shift register instead for the 20-round P_{256} and at each round we take the least significant 4 bits as $RC[r]$. The update function of the 6-bit linear feedback shift register is borrowed from LED [8] and defined as follows. Let $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ be the 6 bits which are initialized to zero. At each round r, $(rc_5, rc_4, rc_3, rc_2, rc_1, rc_0)$ are shifted one position to the left with rc_0 being updated with $rc_5 \oplus rc_4 \oplus 1$. Then, the concatenation of rc_3, rc_2, rc_1 and rc_0 is used as $RC[r]$. Explicitly, $RC = [0x1, 0x3, 0x7, 0xf, 0xf, 0xe, 0xd, 0xb, 0x7, 0xf, 0xe, 0xc, 0x9, 0x3, 0x7, 0xe, 0xd, 0xa, 0x5, 0xb]$.
- AddDomain[d]: as multiple of permutations are needed in the AEAD design, AddDomain[d] xors the domain separator d to all cells of the second column at each round, as depicted in Fig. 1.6.
- SubCells is a nonlinear substitution that applies the PRESENT S-box, as shown below, to each cell of the internal state.

$$S = [0xc, 0x5, 0x6, 0xb, 0x9, 0x0, 0xa, 0xd, 0x3, 0xe, 0xf, 0x8, 0x4, 0x7, 0x1, 0x2]$$

- ShiftRows is a cyclic rotation of i-th row by i bytes to the left, for $i = 0, ..., 7$.
- MixColumnSerial is a multiplication of each column with a matrix M over $GF(2^4)$ by 8 times, where $GF(2^4)$ is defined by the irreducible polynomial $x^4 + x + 1$. The matrix M can be implemented in an extremely compact way and M^8 results in an Maximum Distance Separable (MDS) matrix over $GF(2^4)$.

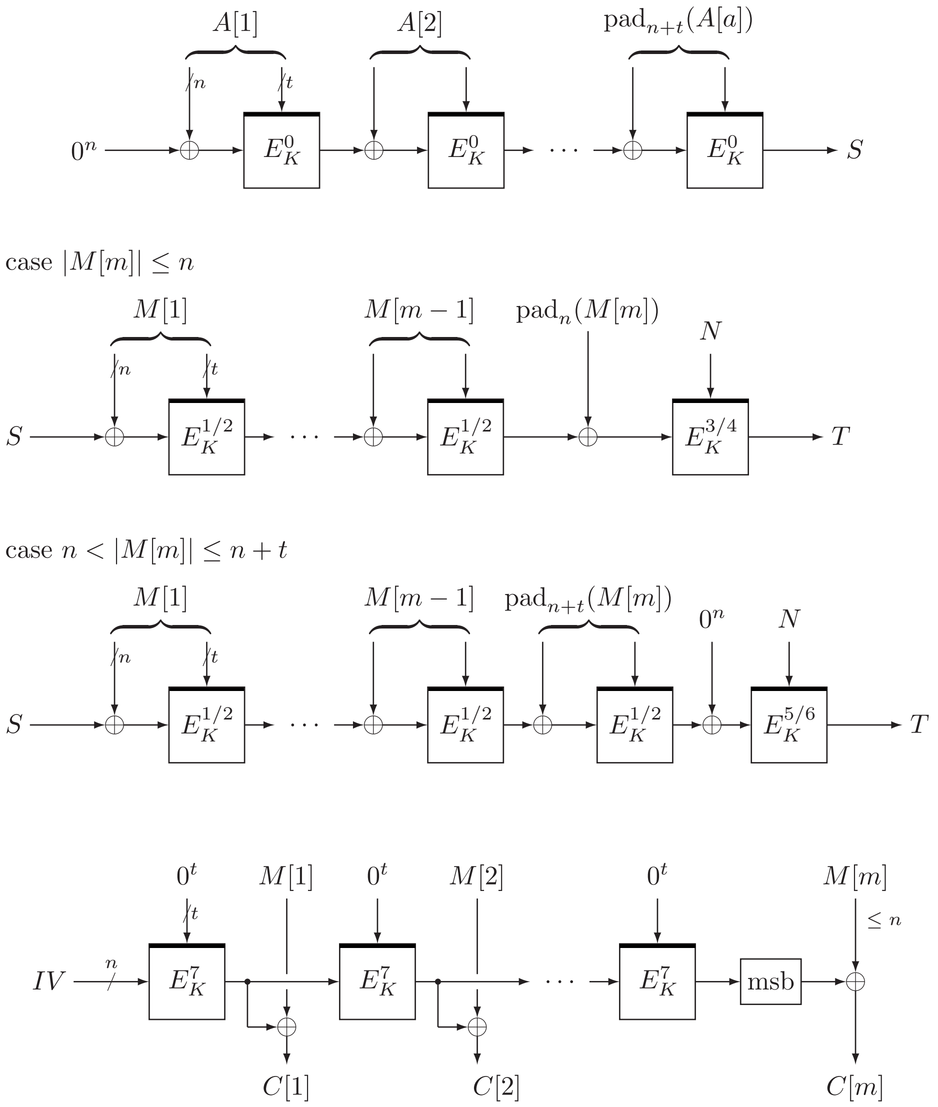

Figure 1.4: The overall structure of SIV scheme. Top: Process of AD A in $\mathcal{F}_K(N, A, M)$. 2nd: Process of a plaintext M in $\mathcal{F}_K(N, A, M)$ for the case $|M[m]| \leq n$. 3rd: Process of M in $\mathcal{F}_K(N, A, M)$ for the case $n < |M[m]| \leq n + t$. Bottom: $C = \mathcal{E}_K^{IV}(M)$. Note that $IV = T$.

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 2 & 4 & 2 & 11 & 2 & 8 & 5 & 6 \end{bmatrix}$$

The original P_{256} is defined to be the round function,

$$\mathsf{RoundFunction[r]} = \mathsf{MixColumnSerial} \circ \mathsf{ShiftRows} \circ \mathsf{SubCells} \circ \mathsf{AddConstant[r]},$$

iterated for $r = 1, \cdots, 12$. To incorporate the domain separator, we define $P_{256}[d]$ with AddDomain[d] so the new round function becomes

$$\mathsf{RoundFunction[r,d]} = \mathsf{MixColumnSerial} \circ \mathsf{ShiftRows} \circ \mathsf{SubCells} \circ \mathsf{AddDomain[d]} \circ \mathsf{AddConstant[r]}.$$

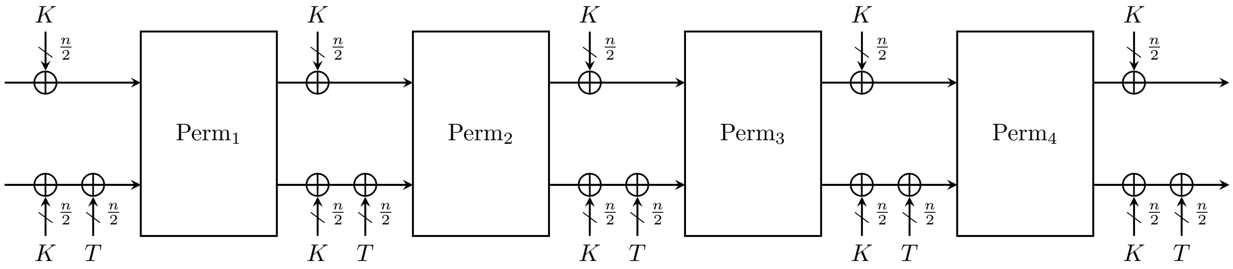

Figure 1.5: Tweakable Even-Mansour construction

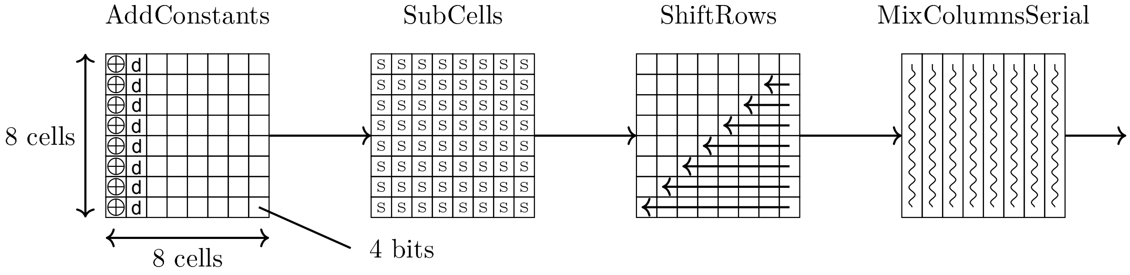

Figure 1.6: Round Function of P_{256} [7], together adding of the domain separator d in the second column.

1.2 Specification of SIV-TEM-PHOTON Family

1.2.1 Specification of TEM-PHOTON

The 4 underlying permutations Perm_i are defined as the RoundFunction[r,d] iterated for $r = 5i-4, 5i-3, \cdots, 5i$ and $i = 1, 2, 3, 4$, as depicted in Fig. 1.7. Here, $|K| = |T| = 128$ bits.

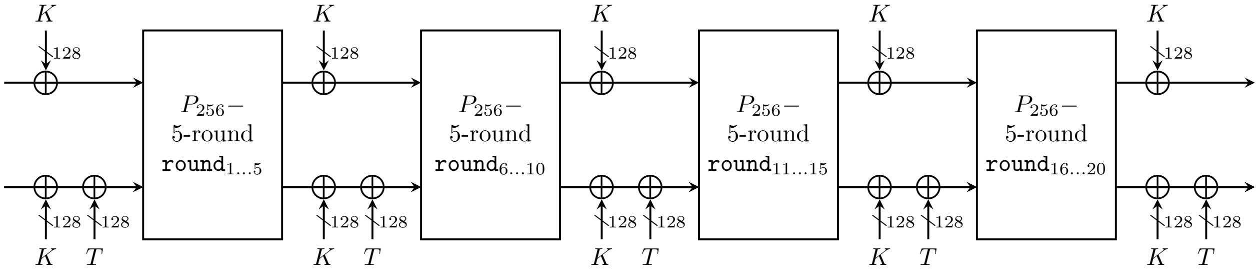

Figure 1.7: Specification of TEM-PHOTON

1.2.2 SIV-TEM-PHOTON-AEAD Authenticated Encryption

The SIV-TEM-PHOTON-AEAD is then the SIV mode, instantiated with TEM-PHOTON defined above. The SIV-TEM-PHOTON-AEAD family consists of only one instance, with the parameter sizes:

- block size $n = 256$ bits,
- key size $k = 128$ bits,
- tag size $|T| = 256$ bits,
- nonce length $|N| = 128$ bits.

It supports the following:

- any bit length of associated data $|A| \geq 0$,

- any bit length of messages $|M| \geq 0$.

Due to the mode, decryption algorithm of the cipher is not necessary.

1.2.3 SIV-TEM-PHOTON-hash Hash Function

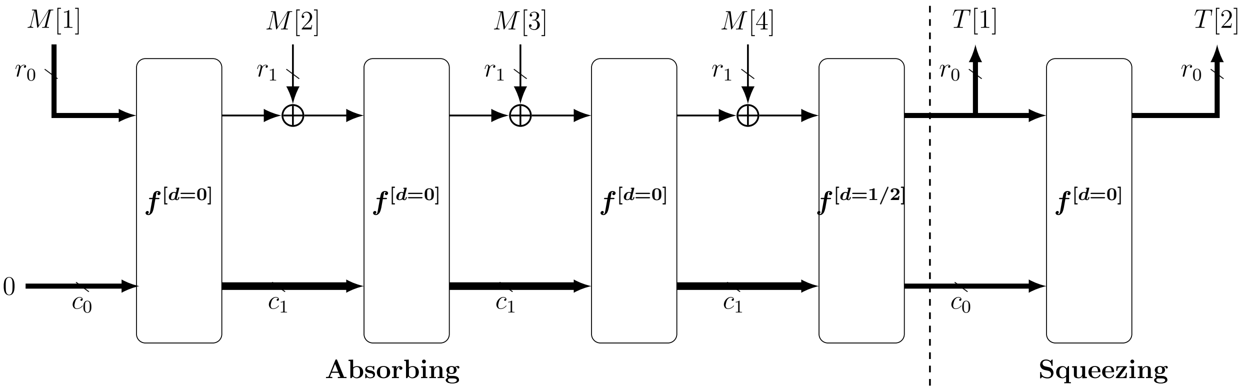

Figure 1.8: SIV-TEM-PHOTON-hash adopts Sponge-like construction, with initial absorbing bitrate r_0, internal absorbing bitrate r_1 and squeezing bitrate r_0.

```
Algorithm SpongeHash[$f^{[0/1/2]}$, pad, $r_0, r_1$, ds]($M$)

Absorption Phase:
$(M[1], \ldots, M[m]) \xleftarrow{r_0, r_1} M$
if $|M| \leq r_0$ then
    $d \leftarrow (|M[m]| = r_0)?1:2$
    $M[m] \leftarrow \mathrm{pad}_{r_0}(M[m])$
else
    $d \leftarrow (|M[m]| = r_1)?1:2$
    $M[m] \leftarrow \mathrm{pad}_{r_1}(M[m])$
$S \leftarrow 0$
for $i = 1$ to $m - 1$
    $S \leftarrow S \oplus (M[i] \mid\mid 0^{|S|-|M[i]|})$
    $S = f^{[0]}(S)$
$S \leftarrow f^{[d]}(S \oplus (M[m] \mid\mid 0^{|S|-|M[m]|}))$

Squeezing Phase:
$T = \mathrm{msb}_{r_0}(S)$
for $i = 1$ to $\lceil \mathtt{ds}/r_0 \rceil - 1$
    $S \leftarrow f^{[0]}(S)$
    $T = T \mid\mid \mathrm{msb}_{r_0}(S)$
return $\mathrm{msb}_{\mathtt{ds}}(T)$
```

Figure 1.9: The definition of our modified Sponge construction.

SIV-TEM-PHOTON-hash adopts the Sponge-like construction (as shown in Fig. 1.8 and 1.9). The difference with Sponge is that the initial absorbing rate and the squeezing rate are larger than the internal absorbing rate. Specifically, in SIV-TEM-PHOTON-hash, both of the initial absorbing bitrate and the squeezing bitrate are r_0, whereas, the internal absorbing bitrate is r_1 and output digest size is ds, *i.e.,*

$$r_0 = 128, \qquad r_1 = 32, \qquad c_0 = 128, \qquad c_1 = 224, \qquad \mathtt{ds} = 256.$$

The underlying permutation in SIV-TEM-PHOTON-hash is the TEM-PHOTON with both the master key K and tweak T set to the constant 0, and we keep the additional input d, and denote the resulted permutation as $f^{[d]}$. $d = 0$ is used for all other places, rather than the last call in the absorption phase. The padding rule follows the same as in the AEAD, i.e., when the last message is of full block (128 bits if $|M| \leq 128$, 32 bits otherwise), no padding is necessary otherwise a bit string of 10^* is padded, and the corresponding d is defined as 1 for full block, and 2 for non-full block.

Chapter 2

Security

2.1 Summary of Expected Security Strength

Attack Model	Time Complexity	Data Complexity
Key Recovery	128 bits	128 bits
Forgery	128 bits	128 bits

Table 2.1: The security claims of SIV-TEM-PHOTON-AEAD.

collision	second-preimage	preimage
112 bits	112 bits	128 bits

Table 2.2: The security claims of SIV-TEM-PHOTON-hash.

2.2 Summary of Known Cryptanalytic Attacks

Differential/Linear cryptanalysis. The underlying TEM cipher is split into four 5-round permutations by the addition of tweak and key (see Fig. 1.7). According to the design strategy of AES-like functions, any consecutive four rounds of P_{256} ensure 81 active S-boxes. Further, upper bounds of the differential probability and linear hull probability of any 4 rounds can be obtained by adapting the work from [12], which are both 2^{-128} as explained in [7]. Therefore, a 5-round P_{256} which certainly activates more S-boxes does not have any differential or linear hull with probability higher than 2^{-128}. More importantly, at least two 5-round permutations of the underlying TEM cipher will be active whether there is a key/tweak difference or not, which gives us much strong confidence in the security against differential/linear attacks.

Internal differential cryptanalysis. The internal difference here is defined to be the difference between the left half and the right half of the internal state. Note that round constants are added only to the first column of the internal state in the round function of P_{256}. That is to say, differences are introduced by AddConstant[r] at each round between the first column and the fifth column of the internal state. Unlike the traditional differential cryptanalysis, no provable bounds on the number of active S-boxes can be obtained for such internal differential trails. Therefore, we lower bound the number of active S-boxes experimentally in the following way. First, let us elaborate a bit more on the setting of attack. In TEM-PHOTON, the 128-bit key is xored to the upper and lower half of the internal state respectively. Additionally, the 128-bit tweak is xored to the lower half of the internal state at the same time. Since it is less realistic to control the difference between the left 64-bit key and the right 64-bit key if the key is generated randomly, we just consider these two parts are the same, i.e., the weak key setting. On the contrary, the tweak difference can be controlled easily and the same tweak difference will be injected every five rounds. Second, we search for optimal internal differential trails of reduced-round P_{256} with constraint programming [6]. However, we only obtain optimal 2-round internal differential trails. When the number of rounds is greater than 2, the search becomes inefficient. Based on the optimal internal differential trails of all possible two consecutive rounds, we obtain the corresponding minimal numbers of active S-boxes. Lastly, the lower bounds on the number of active S-boxes of the four 5-round permutations are obtained using mixed integer linear programming [11] with the bounds on the number of active S-boxes of two consecutive rounds as additional constraints. As a result, the numbers of active S-boxes of the four 5-round permutations are at least 30, 29, 30 and 29 respectively. In total, there are at least 118 active S-boxes which is enough for a 128-bit key. Note that these bounds are not tight.

Existing distinguishers of the PHOTON permutation. Besides the original paper of PHOTON [7], so far there have been two main works on the cryptanalysis of PHOTON in the literature [9, 15], both of which constructed distinguishers of the underlying keyless permutations. The original paper [7] and the work in [9] exploited rebound attacks [10], while the work in [15] studied the division property [14] and then proposed a 12-round zero-sum distinguisher of P_{256} with time complexity 2^{184}, which is the best cryptanalytic result on P_{256} up to date. Since the permutation P_{256} used in SIV-TEM-PHOTON only differs in the operation of adding constants, these distinguishers on original P_{256} still apply (for the underlying permutation in the hashing mode). If it is the case of AEAD with a 128-bit key, similar distinguishers can be constructed by only considering "forward direction" (since no decryption is used), which means the length of the distinguishers will be halved. As can be seen from Tab. 2.3, the length of the distinguishers can be at most 6 rounds.

Number of Rounds	Time (Enc)	Memory (Blocks)	Attack type	Source
8	2^8	2^4	Rebound attack	[7]
9	2^{184}	2^{32}	Rebound attack	[9]
11	2^{119}	-	Division	[15]
12	2^{184}	-	Division	[15]

Table 2.3: Summary of distinguishers on the PHOTON permutation P_{256}

Chapter 3

Design Rationale

Our design goals are summarized as follows.

- The AEAD scheme that is suitable for use lightweight applications.
- Strong security guarantee, based on well analyzed and trusted components and well established mode of operation.
- Address misuse cases of nonce-repetition and release of unverified plaintexts.
- Avoid the use of decryption algorithm of the underlying block cipher, for AEAD decryption.
- The hash function can be easily defined by reusing the components from the AEAD scheme, under a well understood mode, slightly modified for efficiently processing short messages.
- All necessary modifications are kept to be minimum, without affecting the security and lightweightness in hardware/software.

3.1 Choice of SIV

Above goals in mind, we decided to use the SIV mode [13] as our mode of operation. The mode enjoys the provable security in the strong sense of nonce-misuse case, and it also has the provable security in terms of release of unverified plaintexts [1]. The combined OFB mode also enables decryption of the AEAD without the use of decryption algorithm of the underlying tweakable block cipher, which saves the gates required in the hardware implementations and reduces the code size or ROM in software implementations.

Our choice of CBC MAC is to build our scheme on an established standard scheme, yet, in order to gain high performance to process associated data, we decided to use a tweakable block cipher (TBC) as the underlying primitive, a variant of which was first introduced in [4] with underlying component first introduced in [7]. With this approach, the tag generation part of SIV becomes roughly 1.5 times faster per primitive call. More precisely, for AD A and a plaintext M, our mode requires roughly $|A|/(n+t)$ TBC calls to process AD, and $|M|/(n+t)+|M|/n$ calls to process M, where $|M|/(n+t)$ calls are for authentication and $|M|/n$ calls are for encryption. We use OFB mode for its solid provable security guarantee and its small footprint in implementations. Overall, the mode requires such a small amount of gates to implement that the overall amount of gates required by the AEAD design is almost the same as that by the underlying P_{256} permutation, when the keys and tweak inputs are provided by the external controller.

The security bound is the standard birthday bound of the form $O(\sigma^2/2^n)$, where σ denotes the total number of blocks in the security game. With the application for hashing in mind, we adopt a block size of $n = 256$, which gives a solid security bound for any lightweight applications.

3.2 Choice of TEM

In order to obtain a TBC, we adopt TEM for our construction. This construction avoids the need for key/tweak scheduling, and this allows saving logic gates in hardware implementations. Besides, the simplicity of the construction allows an extensive security analysis. We use 4 iterations of round-reduced P_{256} to be conservative, as constructions with 3 iterations already offers the birthday security [4], and we further reduce the tweak size by half to reduce the freedom degree attackers might have. We appropriately use round constants from the design of LED to avoid slide attacks, which effectively act as the domain separation in the random permutation model.

3.3 Choice of PHOTON

First of all, AES-like permutations offer much confidence in security as one can leverage previous cryptanalysis works done on AES. It is known that AES-like permutations allow to derive simple proofs on the number of active S-boxes of any consecutive four rounds. Specifically, at least $9^2 = 81$ S-boxes will be active for any four rounds of P_{256} and this bound is tight. Secondly, the PHOTON permutation is lightweight and the PHOTON hash function has been standardized by ISO since 2016. Notably, the PHOTON permutation P_{256} can be implemented with less than 1736 GEs in an extremely compact way. Lastly, the PHOTON permutation adopts a simple round constant generation, which can be extended naturally to more rounds without additional costs.

Chapter 4

Performance

4.1 Hardware Performance

The hardware performance and implementation cost is expected to be an advantage of SIV-TEM-PHOTON. The area cost can be very small considering the following two points: 1. the mode SIV costs little on top of the costs of the underlying block cipher; 2. the underlying block cipher TEM-PHOTON is tweaking on P_{256} which is one of the most compact primitives with the same dimension.

To estimate the area cost of the hardware implementation of SIV-TEM-PHOTON, we use the available results on that of the hash function PHOTON-224/32/32, which also uses P_{256} as its underlying permutation. PHOTON-224/32/32 adopts Sponge construction with in-/output bitrate 32/32. Considering that the Sponge construction also costs little on top of the costs of the underlying permutation, it is reasonable to base on the area of the hardware implementation of PHOTON-224/32/32 to estimate that of SIV-TEM-PHOTON.

According to [7], as for serial ASIC implementations of PHOTON-224/32/32 using the standard cell library UMCL18G212T3 (with data path $s = 4$, which is the size of cells in the state), when target at minimizing area, it costs 1736 GEs and the latency of the underlying permutation is 1716 clock cycles; when target at minimizing latency, it costs 2786 GEs and the latency of the underlying permutation is 204 clock cycles.

Comparing implementations of SIV-TEM-PHOTON with that of PHOTON-224/32/32, additional costs of area comes from the storage for key, tweak, domain separator (and the XOR gates for addition of them). However, since key bits and tweak bits are used without schedule in TEM-PHOTON, in the case where they can be sent multiple times by the external provider, local storage can be saved. In serial implementations with small data path (*e.g.,* $s = 4$), adding key, tweak, and larger message blocks can be serialized and thus require limited number of additional XOR gates (suppose 1-bit XOR gates cost 2.67 GEs, for data path being 4, additional XOR gates cost about $(4 + 4 + 3) \times 2.67 \approx 30$ GEs).

Hence, in the case where key and tweak do not need to be stored locally, we estimate the area of implementations of SIV-TEM-PHOTON is close to that of PHOTON-224/32/32.

When key and tweak has to be stored locally, they can be stored using 256 regular 1-bit flip-flops. Suppose one 1-bit regular flip-flops costs 4.67 GEs, the additional area costed can be estimated as $256 \times 4.67 = 1195.52$ GEs.

4.2 Software Performance

The software performance in general purpose processors is a disadvantage of SIV-TEM-PHOTON. According to [7], the software performance of PHOTON-224/32/32 is about 227 cycles per byte for long messages in an Intel(R) Core(TM) i7 CPU. In SIV-TEM-PHOTON-AEAD, during the authentication phase, 1.5-block-size messages are processed per call of the underlying permutation. During the encryption phase, full-block-size messages are processed per call of the underlying permutation. In the case that message and authenticated data are with similar length, it can be estimated as processing $(1.5 + 1)/2$-block-size message per call. The block size is 256-bit in SIV-TEM-PHOTON-AEAD, which is 8 times of the input bitrate of PHOTON-224/32/32. TEM-PHOTON has 20 rounds while PHOTON-224/32/32 has only 12 rounds. Thus, we estimate that for long messages, the performance of SIV-TEM-PHOTON-AEAD be $(2/(1.5+1)) * (20/12) * 227 * (1/8) \approx$

37.9 cycles per bytes. For SIV-TEM-PHOTON-hash which has larger initial absorbing bitrate and lager squeezing bitrate, the number of cycles per byte is estimated to be less than 227.

However, considering the targeted usage scenario is on constrained devices instead of high-end CPUs, we focus on the software implementation and performance on micro-controllers which should not be a limitation of SIV-TEM-PHOTON.

According to a report on the implementation and performance evaluation of Hash functions in ATtiny devices [2], the code size of PHOTON-256/32/32 (which uses PHOTON P_{288} as its underlying permutation with the 8-bit S-box of AES) is 1244 bytes, the RAM requirement is 78 bytes. The code size of PHOTON-160/36/36 (which uses PHOTON-P_{196} as its underlying permutation with the 4-bit S-box of PRESENT, which is the same as the one used in TEM-PHOTON) is 764 bytes, the RAM requirement is 50 bytes. Considering the performance of PHOTON-224/32/32 should lie in-between that of these two primitives, following the implementation methods in [2], the code size for TEM-PHOTON should be at the range of 764 $\sim$ 1244 bytes, and the SRAM requirement should be at the range of 50 $\sim$ 78 bytes. Besides, our primary bit-sliced implementation (bit-slicing within a single state) of the underlying TEM-PHOTON requires 768 bytes ROM (720 for code and 48 for data, including codes for bit-slicing) and 32 bytes RAM (exclude those used for key/nonce/messages/outputs).

Considering our mode SIV can be implemented with small number of additional instructions and small number of constants, we expect that in 8-bit AVR devices, the implementations of SIV-TEM-PHOTON require less than 1000 bytes ROM and less than 64 bytes SRAM.

We note that, in SIV-TEM-PHOTON-AEAD, although the message are processed twice to achieve the features of SIV, they are processed in large blocks ((256+128)- or 256-bit each) per call of the underlying TEM-PHOTON. Thus, we expect that the throughput should not be a limitation.

Chapter 5

References

5.1 Reference/Third-party Analysis on PHOTON

Cryptanalysis:

- Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Improved Rebound Attack on the Finalist Grøstl. In Anne Canteaut, editor, Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers, volume 7549 of Lecture Notes in Computer Science, pages 110–126. Springer, 2012.
- Qingju Wang, Lorenzo Grassi, and Christian Rechberger. Zero-Sum Partitions of PHOTON Permutations. In Nigel P. Smart, editor, Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings, volume 10808 of Lecture Notes in Computer Science, pages 279–299. Springer, 2018.
- Chia-Yu Lu, You-Wei Lin, Shang-Ming Jen, and Jar-Ferr Yang. Cryptanalysis on PHOTON Hash Function Using Cube Attack. In 2012 International Conference on Information Security and Intelligent Control, pages 278–281, Aug 2012.

Implementations:

- N. Nalla Anandakumar, Thomas Peyrin, and Axel Poschmann. A Very Compact FPGA Implementation of LED and PHOTON. In Willi Meier and Debdeep Mukhopadhyay, editors, Progress in Cryptology - INDOCRYPT 2014 - 15th International Conference on Cryptology in India, New Delhi, India, December 14-17, 2014, Proceedings, volume 8885 of Lecture Notes in Computer Science, pages 304–321. Springer, 2014.
- Josep Balasch, Baris Ege, Thomas Eisenbarth, Benoît Gérard, Zheng Gong, Tim Güneysu, Stefan Heyse, Stéphanie Kerckhof, François Koeune, Thomas Plos, Thomas Pöppelmann, Francesco Regazzoni, François-Xavier Standaert, Gilles Van Assche, Ronny Van Keer, Loïc van Oldeneel tot Oldenzeel, and Ingo von Maurich. Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices. In Stefan Mangard, editor, Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers, volume 7771 of Lecture Notes in Computer Science, pages 158–172. Springer, 2012.
- Pavan Kumar Malka: Compact Hardware Implementation of PHOTON Hash Function in FPGA (2011). `http://ece.gmu.edu/coursewebpages/ECE/ECE646/F11/project/F11presentations/Pavan.pdf`

Side-channel attacks:

- Wei Li, Linfeng Liao, Dawu Gu, Chenyu Ge, Zhiyong Gao, Zhihong Zhou, Zheng Guo, Ya Liu and Zhiqiang Liu, Security Analysis of the PHOTON Lightweight Cryptosystem in the Wireless Body Area Network, KSII Transactions on Internet and Information Systems, vol. 12, no. 1, pp. 476-496, 2018. DOI: 10.3837/tiis.2018.01.023

Acknowledgements

The submitters would like to thank Kazuhiko Minematsu of NEC Corporation for various discussions in the early design stage of our submission.

Bibliography

[1] Elena Andreeva, Andrey Bogdanov, Atul Luykx, Bart Mennink, Nicky Mouha, and Kan Yasuda. How to securely release unverified plaintext in authenticated encryption. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 105–125, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany.

[2] Josep Balasch, Baris Ege, Thomas Eisenbarth, Benoît Gérard, Zheng Gong, Tim Güneysu, Stefan Heyse, Stéphanie Kerckhof, François Koeune, Thomas Plos, Thomas Pöppelmann, Francesco Regazzoni, François-Xavier Standaert, Gilles Van Assche, Ronny Van Keer, Loïc van Oldeneel tot Oldenzeel, and Ingo von Maurich. Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, volume 7771 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 2012.

[3] Shan Chen and John P. Steinberger. Tight security bounds for key-alternating ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 327–350, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.

[4] Benoit Cogliati, Rodolphe Lampe, and Yannick Seurin. Tweaking Even-Mansour ciphers. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 189–208, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[5] Shimon Even and Yishay Mansour. A Construction of a Cipher from a Single Pseudorandom Permutation. *J. Cryptology*, 10(3):151–162, 1997.

[6] David Gerault, Marine Minier, and Christine Solnon. Constraint Programming Models for Chosen Key Differential Cryptanalysis. In Michel Rueher, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 584–601. Springer, 2016.

[7] Jian Guo, Thomas Peyrin, and Axel Poschmann. The PHOTON Family of Lightweight Hash Functions. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 222–239. Springer, 2011.

[8] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *LNCS*, pages 326–341, Nara, Japan, September 28 – October 1, 2011. Springer, Heidelberg, Germany.

[9] Jérémy Jean, María Naya-Plasencia, and Thomas Peyrin. Improved Rebound Attack on the Finalist Grøstl. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 110–126. Springer, 2012.

[10] Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2009.

[11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *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*, pages 57–76. Springer, 2011.

[12] Sangwoo Park, Soo Hak Sung, Sangjin Lee, and Jongin Lim. Improving the Upper Bound on the Maximum Differential and the Maximum Linear Hull Probability for SPN Structures and AES. In Thomas Johansson, editor, *Fast Software Encryption, 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003, Revised Papers*, volume 2887 of *Lecture Notes in Computer Science*, pages 247–260. Springer, 2003.

[13] Phillip Rogaway and Thomas Shrimpton. A Provable-Security Treatment of the Key-Wrap Problem. In *EUROCRYPT*, pages 373–390, 2006.

[14] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 287–314. Springer, 2015.

[15] Qingju Wang, Lorenzo Grassi, and Christian Rechberger. Zero-Sum Partitions of PHOTON Permutations. In Nigel P. Smart, editor, *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*, volume 10808 of *Lecture Notes in Computer Science*, pages 279–299. Springer, 2018.