Malaysian Journal of Mathematical Sciences 15(S) December: 1–23 (2021)

Malaysian Journal of Mathematical Sciences

Journal homepage: https://einspem.upm.edu.my/journal

The CiliPadi Family of Lightweight Authenticated Encryption, v1.2

Z'aba, M. R.[1,*], Jamil, N.[2], Rohmad, M. S.[3], Rani, H. A.[4], and Shamsuddin, S.[4]

[1]*Department of Computer System and Technology, Faculty of Computer Science and Information Technology, Universiti Malaya, Malaysia*
[2]*College of Computing and Informatics, Universiti Tenaga Nasional, Malaysia*
[3]*Faculty of Electrical Engineering, Universiti Teknologi MARA, Malaysia*
[4]*CyberSecurity Malaysia*

E-mail: reza.zaba@um.edu.my
**Corresponding author*

Received: 8 January 2020
Accepted: 4 August 2021

Abstract

This article describes the specification and analysis of the CiliPadi family of lightweight authenticated encryption v1.2. An earlier version, dubbed v1.0, was accepted as one of the Round 1 candidates in the US NIST lightweight cryptography project. CiliPadi is designed based on the Sponge construction which is also used in the SHA-3 hash function. CiliPadi supports 128- and 256-bit keys and is offered in four variants or flavours. The flavours differ in the length of tag, message block and the number of rounds of the internal permutation.

Keywords: authenticated encryption; sponge construction; lightweight cryptography; symmetric encryption; generalized Feistel network.

1 Introduction

Traditional encryption such as stream and block ciphers only assures confidentiality of data. To additionally ensure integrity and authenticity, authenticated encryption (AE) schemes are used. Such schemes can be built using the generic composition approach [3], i.e. by interweaving individual encryption and message authentication code (MAC) schemes. The three generic composition approaches are: Encrypt-and-MAC, MAC-then-encrypt and Encrypt-then-MAC.

However, due to the intricate nature of securely combining these schemes, a dedicated primitive that intrinsically provides confidentiality, integrity and authenticity is required. For instance, Bellare and Namprempre [3] have shown that only Encrypt-then-MAC satisfies all the privacy and integrity security notion when the underlying MAC is assumed to be strongly unforgeable. Vaudenay [30] reports that when padding is performed in between encryption and the MAC operation, the scheme is susceptible to attacks. Bellare, Kohno and Namprempre [2] find out that, due to the use of predictable IVs in the AE scheme of the SSH protocol, an attacker is able to obtain some information about previously encrypted messages. Paterson and Watson [26] show that improper handling of padding in the generic composition approach results in an insecure scheme.

Early proposals that address this problem appear in the early 2000s. Examples include the related plaintext chaining (RPC) by Katz and Yung [23] and the integrity aware CBC (IACBC) and integrity aware paralellizability mode (IAPM) by Jutla [22]. Rogaway et al. [27] then introduces the OCB mode, which enhances the IAPM. Gligor and Donescu [18] propose the XCBC and XECB construction which make use of XOR to provide authentication of messages using a block cipher in single-pass. These dedicated AE proposals are derived based on either a block cipher, or a permutation called *sponge* [7].

The United States (US) National Institute of Standards and Technology (NIST) has been influential in shaping international cryptographic standards, and AE is no exception. In 2018, the NIST issued a call-for-algorithms for the NIST lightweight cryptography (LwC) standardisation project. The project specifically focuses on *lightweight* AE algorithms, i.e. algorithms that are suitable to be implemented in devices where resources such as area, memory and power, are limited. Such resource-constrained devices typically include the Internet-of-Things (IoTs), which are currently gaining significant attention.

CiliPadi is a family of lightweight AE algorithms. An earlier version of CiliPadi [31], referred to as v1.0, was submitted to the NIST LwC project on March 2019. It was accepted as one of the 56 candidates in the Round 1 phase of the project. In August 2019, the Round 2 candidates were announced and CiliPadi did not make the cut. This is due to a length extension attack by Bagheri [1, 29]. This article describes an updated version of CiliPadi labelled as version 1.2 (v1.2). This updated version is resistant against the length-extension attack by including a proper padding scheme for the inputs described later in Section 3.3.

CiliPadi is based on the MonkeyDuplex construction [5] which evolves from the original Sponge proposed in 2007 [7]. The versatile Sponge construction has been extensively scrutinized and deployed in numerous hash functions and AE proposals (e.g. Keccak [4], PHOTON [20], Fides [8] and Ascon [16]. There are four main flavours of CiliPadi, i.e. CiliPadi-Mild, CiliPadi-Medium, CiliPadi-Hot and CiliPadi-ExtraHot. The flavours differ in the length of tag, message block and the number of rounds of the internal permutation. The permutation function makes use of an unkeyed 2-round of the lightweight block cipher `LED` [21] as the F-function in a Type-II generalized feistel network (GFN) [32]. This is similar to the Simpira v2 permutation framework introduced by Gueron and Mouha [19].

This article is organised as follows. Section 2 describes the preliminaries to understand the subsequent sections in this article. The specification of CiliPadi is provided in Section 3. The rationale regarding the design choices of CiliPadi is explained in Section 4. Section 5 outlines the performance of CiliPadi simulated on hardware. The security analysis of CiliPadi is detailed in Section 6. The known strengths and weaknesses of CiliPadi are given in Section 7.

2 Preliminaries

This section introduces the notation and other preliminaries as basis to understand subsequent sections.

2.1 Notations

The notation used in this paper is given in Table 1.

Table 1: Notation.

	Description
K, N, T	The secret key, nonce, and tag, respectively
$X\|Y$	The concatenation of bit strings X and Y
A	The associated data where $A = A_1\|\dots\|A_s$
M	The message where $M = M_1\|\dots\|M_t$
C	The ciphertext where $C = C_1\|\dots\|C_t\|T$ and T is the tag
$\lvert X\rvert$	The length of X in bits
S	The internal state where $S = S_r\|S_c$, $r = \lvert S_r\rvert$ and $c = \lvert S_c\rvert$
n	The length of the internal state S, i.e. $n = \lvert S\rvert = r + c$
$\mathtt{0}_2^x, \mathtt{1}_2^x$	An all-zeros and all-ones binary string of x bits, respectively
$\lceil X\rceil^i$	The first i bits (or leftmost bits) of X
$(X_1\|\dots\|X_x) \xleftarrow{y} X$	The parsing of the bit string X into x equally-sized y-bit strings

A number written in typewriter font is always treated as a 4-bit hexadecimal value, i.e. `0`, `1`, ... `f`. If the value is subscripted with '2', as shown in Table 1 then it is treated as a 1-bit value, which applies only for `0` and `1`.

2.2 Mode of Operation

The CiliPadi$[n, r, a, b]$ mode of operation is based on the MonkeyDuplex construction [5] and depicted in Figure 1. It consists of four phases: initialization, associated data authentication, message encryption/decryption, and finalization. The state length is n bits initialized with the value of the secret key K and nonce N. The bitrate is r bits and the capacity is $c = n - r$ bits. The

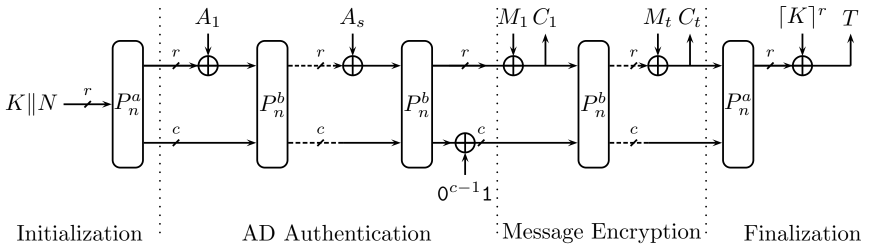

Figure 1: CiliPadi mode of operation.

permutation for the initialization and finalization phases has a rounds while the permutation for the associated data and message encryption/decryption phases has b rounds where $b > a$.

3 Specification

This section formally describes the specification of CiliPadi.

3.1 Parameters

The CiliPadi$[n, r, a, b]$ family of authentiated encryption (AE) scheme consists of configurable parameters. For the purpose of evaluation, we propose four flavours of CiliPadi which is listed in Table 2 according to increasing level of security. They are CiliPadi-Mild, CiliPadi-Medium, CiliPadi-Hot and CiliPadi-ExtraHot. The lengths stated in the table are all in bits.

Table 2: CiliPadi parameters where the primary member is CiliPadi-Mild.

CiliPadi	Algorithm	Length of				No. of rounds	
	$[n, r, a, b]$	Key	Nonce	Tag	Block	P_n^a	P_n^b
Mild	$[256, 64, 18, 16]$	128	128	64	64	18	16
Medium	$[256, 96, 20, 18]$	128	128	96	96	20	18
Hot	$[384, 96, 18, 16]$	256	128	96	96	18	16
ExtraHot	$[384, 128, 20, 18]$	256	128	128	128	20	18

Formally, the CiliPadi family of AE accepts a k-bit secret key K and a 128-bit nonce N. These values become the initial value of the n-bit internal state $S = K\|N$. The state is then updated by the permutation P_n^a. If the sr-bit associated data $A = A_1\|\ldots\|A_s$ is non-empty, it will be subsequently processed, along with the internal state, by the associated data authentication phase. Encryption takes the padded message $M = M_1\|\ldots\|M_t$ and outputs the ciphertext $C = C_1\|\ldots\|C_t$ and tag T where $|M_i| = |C_i| = T = r$ bits. Decryption takes the ciphertext C and tag T and outputs the original message M if and only if C is authentic, else it outputs $\perp$. An algorithmic description of the components of CiliPadi is given in Figure 2, while the high-level overview is provided in Figure 3. The descriptions are provided in the following sections.

```
Proc Init$(K, N)$
$S \leftarrow K \| N$
return $S$
```

```
Proc AD$(S, A)$
for $i = 1, \ldots, s$ do
    $S \leftarrow P_n^b((S_r \oplus A_i) \| S_c)$
end
$S \leftarrow (S_r \| S_c \oplus (\mathtt{0}_2^{c-1} \| \mathtt{1}))$
return $S$
```

```
Proc Finalization$(S)$
$S \leftarrow P_n^a(S)$
$T \leftarrow \lceil S \rceil^k \oplus K$
return $T$
```

```
Proc MEnc$(S, M)$
for $i = 1, \ldots, t-1$ do
    $C_i \leftarrow S_r \oplus M_i$
    $S \leftarrow P_n^b(C_i \| S_c)$
end
$C_t \leftarrow S_r \oplus M_t$
$S \leftarrow (C_t \| S_c)$
return $(S, C)$
```

```
Proc MDec$(S, C)$
for $i = 1, \ldots, t-1$ do
    $M_i \leftarrow S_r \oplus C_i$
    $S \leftarrow P_n^b(C_i \| S_c)$
end
$M_t \leftarrow S_r \oplus C_t$
$S \leftarrow (C_t \| S_c)$
return $(S, M)$
```

```
Proc $P_{256}^r(S)$
$(X_1 \| \ldots \| X_4) \xleftarrow{64} S$
for $i = 1, \ldots, r$ do
    $Y_1 \leftarrow F_1(X_1) \oplus X_2$
    $Y_2 \leftarrow X_3$
    $Y_3 \leftarrow F_2(X_3) \oplus X_4$
    $Y_4 \leftarrow X_1$
    $X \leftarrow Y$
end
$S \leftarrow X$
return $S$
```

```
Proc $P_{384}^r(S)$
$(X_1 \| \ldots \| X_6) \xleftarrow{64} S$
for $i = 1, \ldots, r$ do
    $Y_1 \leftarrow F_1(X_1) \oplus X_2$
    $Y_2 \leftarrow X_3$
    $Y_3 \leftarrow F_2(X_5) \oplus X_6$
    $Y_4 \leftarrow X_1$
    $Y_5 \leftarrow F_3(X_3) \oplus X_4$
    $Y_6 \leftarrow X_5$
    $X \leftarrow Y$
end
$S \leftarrow X$
return $S$
```

```
Proc $F_l^i(X)$
$(x_1 \| \ldots \| x_4) \xleftarrow{16} X$
$(w_1 \| w_2) \xleftarrow{2} l$
$(z_1 \| z_2) \xleftarrow{3} rc[i]$
$x_1 \leftarrow (\mathtt{0}_2^2 \| w_1 \| \mathtt{0}_2 \| z_1 \| \mathtt{0}_2^8)$
$x_2 \leftarrow (\mathtt{0}_2^2 \| w_2 \| \mathtt{0}_2 \| z_2 \| \mathtt{0}_2^8)$
$x_3 \leftarrow (\mathtt{2} \| \mathtt{0}_2 \| z_1 \| \mathtt{0}_2^8)$
$x_4 \leftarrow (\mathtt{3} \| \mathtt{0}_2 \| z_2 \| \mathtt{0}_2^8)$
$RC \leftarrow (x_1 \| x_2 \| x_3 \| x_4)$
$X \leftarrow \mathrm{LED1r}(X, RC)$
$X \leftarrow \mathrm{LED1r}(X, \mathtt{0}_2^{64})$
return $X$
```

```
Proc LED1r$(X, RC)$
$X \leftarrow \mathtt{AC}(X, RC)$
$X \leftarrow \mathtt{SC}(X)$
$X \leftarrow \mathtt{SR}(X)$
$X \leftarrow \mathtt{MCS}(X)$
return $X$
```

Figure 2: Components of CiliPadi.

```
Proc Encrypt($K, N, A, M$)
1  $S \leftarrow$ Init($K, N$)
2  if $|A| > 0$ then
3      $A \leftarrow$ Pad($A$)
4      $S \leftarrow$ AD($S, A$)
5  end
6  $M \leftarrow$ Pad($M$)
7  $(S, C) \leftarrow$ MEnc($S, M$)
8  $T \leftarrow$ Finalization($S$)
9  return $(C, T)$
```

```
Proc Decrypt($K, N, A, C, T$)
1  $S \leftarrow$ Init($K, N$)
2  if $|A| > 0$ then
3      $S \leftarrow$ AD($S, A$)
4  end
5  $(S, M) \leftarrow$ MDec($S, C$)
6  $T' \leftarrow$ Finalization($S$)
7  if $(T = T')$ then
8      return $M$
9  else
10     return $\perp$
11 end
```

Figure 3: High-level overview of the encryption and decryption of CiliPadi.

3.2 Initialization Phase

The n-bit state S is initialized with the value of the k-bit key followed by the 128-bit nonce N. The internal state S is initialized as

$$S = K \| N.$$

Note that the nonce must never be repeated to encrypt different messages using the same secret key. The internal state can also be viewed as the concatenation of the r-bit rate S_r and c-bit capacity S_c parts, i.e. $S = S_r \| S_c$. The state is then processed by the n-bit permutation P_n^a, which is described later. The output of this phase is fed to the associated data authentication phase, if the associated data is non-empty.

3.3 Padding

Both the associated data and message blocks are individually padded until its entire length is a multiple of r bits. Note that the padding is performed even though the block is already a multiple of r bits. In such a case, an extra block is appended. Padding is performed by adding a bit 1, and then as many zero bits as necessary until the padded data is in multiple of r bits. If the length of the last block is $r - 1$ bits, then only bit 1 is added. Note that this padding scheme is the main difference between CiliPadi v1.2 and v1.0, which was accepted as one of the Round 1 candidates in the US NIST Lightweight cryptography project. In CiliPadi v1.0, padding is added individually to the associated data and message blocks only if their lengths are not a multiple of r bits. This has led to the length extension attack by Bagheri [1, 29]. By including padding even if the lengths of those inputs are already a multiple of r bits, CiliPadi v1.2 is secure against such attack.

3.4 Associated Data Authentication Phase

If the associated data $A = A_1 \| \ldots \| A_s$ is non-empty, then A_1 is XORed with the inner state S_r. The state S is then updated by the permutation P_n^b. This process is repeated for A_i $(i = 1, \ldots, s)$ until all associated data blocks are processed:

$$S \leftarrow P_n^b((S_r \oplus A_i) \| S_c).$$

After the last associated data is processed, the outer state S_c is XORed with the binary string $\mathtt{0}_2^{c-1}\|\mathtt{1}_2$ to denote the completion of the associated data phase:

$$S \leftarrow (S_r \| S_c \oplus (\mathtt{0}_2^{c-1}\|\mathtt{1}_2)).$$

The output of this phase is fed to either the message encryption or decryption phase, described next.

3.5 Message Encryption Phase

There are two main inputs for this phase. The first comes from either the initialization (if the associated data is empty) or associated data authentication phase. The second input comes from the padded message $M = M_1\|\ldots\|M_t$. The current inner state S_r is first XORed with the first message block M_1 to produce the ciphertext block C_1. If there are more message block available, then this process is repeated until all message blocks are processed, except for the last block where the permutation P_n^b is not applied:

$$S \leftarrow \begin{cases} P_n^b((S_r \oplus M_i)\|S_c), & \text{for } \ i = 1, \ldots, t-1, \\ (S_r \oplus M_i)\|S_c, & \text{for } \ i = t. \end{cases}$$

The i-th ciphertext block is extracted from the current state prior to the execution of P_n^b as $C_i = S_r \oplus M_i$. The output of this phase is fed into the finalization phase.

3.6 Message Decryption Phase

The decryption phase is almost identical to encryption. It receives two inputs. The first is the outcome of either the initialization or associated data authentication phase. The second input is the ciphertext $C = C_1\|\ldots\|C_t$. The first ciphertext block assumes the value of S_r and then the state S is updated by P_n^b. This process is repeated until the second last ciphertext block. The last ciphertext block is not processed by P_n^b.

$$S \leftarrow \begin{cases} P_n^b(C_i\|S_c), & \text{for } \ i = 1, \ldots, t-1, \\ C_i\|S_c, & \text{for } \ i = t. \end{cases}$$

The corresponding message block is only released when the tag is verified, i.e. only after succesful execution of the finalization phase. The i-th message block is extracted from the current state prior to the execution of the permutation P_n^b, i.e. $M_i = S_r \oplus C_i$.

3.7 Finalization Phase

This phase updates the internal state S to output a single r-bit tag. After the state has been processed by P_n^a, S_r is XORed with the first r bits of the key K. The tag T is the result of this XOR as follows.

$$S \leftarrow P_n^a(S_r\|S_c),$$
$$T \leftarrow S_r \oplus \lceil K \rceil^r.$$

When decrypting ciphertext, the original message M will only be released if the computed tag above matches the one supplied by the sending party.

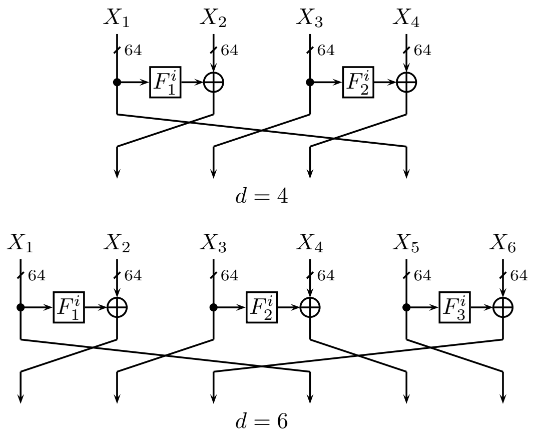

Figure 4: Type-II GFN employed in CiliPadi for $d = 4$ and $d = 6$.

3.8 The Permutation Function P

The permutation function P makes use of an unkeyed 2-round[1] of the lightweight block cipher LED [21] as the round- and line- dependent F-function in a Type-II generalized feistel network (GFN) [32]. We refer a Type-II GFN that accepts d input sub-blocks as a d-line Type-II GFN. For CiliPadi, d is an even number and each line is of length n/d bits. There are $d/2$ F functions in each round that accepts input from odd-numbered lines. Let $X_1 \| \ldots \| X_d$ denote the input lines. They are updated by the F-function in the i-th round as follows.

$$\begin{aligned} X_j &\leftarrow X_j && \text{for } \ j = 1, 3, \ldots, d-1, \\ X_j &\leftarrow X_j \oplus F^i_{j/2}(X_{j-1}) && \text{for } \ j = 2, 4, \ldots, d. \end{aligned}$$

After the above transformations, the lines are shuffled by the permutation function π before being used as input to the next round. For instance, the shuffle $\pi = \{2, 3, 4, 1\}$ means that the first input line is mapped to the second output line, the second input line to the third output line, and so forth. The same shuffle π is used in both the message encryption and decryption phases. For CiliPadi, the shuffling[2] used are given in Table 3 and depicted in Figure 4 for $d = 4$ and $d = 6$.

Table 3: Shuffling used in the Type-II GFN.

Input length n (in bits)	Number of lines d	Shuffle π
128	2	$\{2, 1\}$
256	4	$\{4, 1, 2, 3\}$
384	6	$\{4, 1, 2, 5, 6, 3\}$
512	8	$\{4, 1, 2, 5, 8, 3, 6, 7\}$

[1]This refers to a full 2 rounds where no operation is omitted in the last (second) round.
[2]Note that in [28], the index starts with 0, ours start with 1.

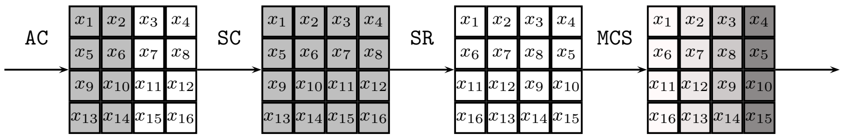
AC
SC
SR
MCS
x_1 x_2 x_3 x_4
x_5 x_6 x_7 x_8
x_9 x_{10} x_{11} x_{12}
x_{13} x_{14} x_{15} x_{16}
x_6 x_7 x_8 x_5
x_{11} x_{12} x_9 x_{10}
x_{16} x_{13} x_{14} x_{15}

Figure 5: A single round of LED.

3.9 The F function

As mentioned earlier, the round- and line- dependent F function is an unkeyed 2-round of the LED [21] block cipher where no operation is omitted in the last (i.e. second) LED round. A single LED round[3] consists of the following four operations, depicted in Figure 5, applied in sequence to the 64-bit input: `AddConstants`, `SubCells`, `ShiftRows` and `MixColumnsSerial`. The input to LED is 64 bits partitioned into 16 4-bit cells. Let $x = x_1\|\ldots\|x_{16}$ denote this input which can be depicted as a 4×4 matrix which is entered row-wise as follows.

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{bmatrix}.$$

In AES, the input is entered column-wise.

3.9.1 AddConstants (AC)

Initialize a 6-bit round constant $rc_1\|\ldots\|rc_6$ set to all zeros. In each round of the permutation, the constant is updated by shifting its value 1 bit to the left. The new value for rc_6 is taken as $rc_1 \oplus rc_2 \oplus 1$. We also add the F-function number encoded as a 4-bit value $l = l_1\|\ldots\|l_4$ as one of the parameters in the constant value. This is to ensure that every F-function has different constant value.

$$\begin{bmatrix} l_1\|l_2 & rc_1\|rc_2\|rc_3 & 0 & 0 \\ l_3\|l_4 & rc_4\|rc_5\|rc_6 & 0 & 0 \\ 2 & rc_1\|rc_2\|rc_3 & 0 & 0 \\ 3 & rc_4\|rc_5\|rc_6 & 0 & 0 \end{bmatrix}.$$

The above matrix is XORed to the current value of the state in the first round of LED in the F-function. The constants for the second round of LED are set to all-zeros. The complete values of the round constants are given in Table 4.

Table 4: The values for the second column of the round constant state matrix used in the first LED-round of all F-functions. The round constants for the second LED-round are set to all-zeros. The first column of the round constant matrix is $(0,1,2,3)$ for F_1^i, $(0,2,2,3)$ for F_2^j, and $(0,3,2,3)$ for F_3^j.

Rnd.	Value	Rnd.	Value	Rnd.	Value	Rnd.	Value
1	$(0,1,0,1)$	6	$(7,6,7,6)$	11	$(3,6,3,6)$	16	$(1,6,1,6)$
2	$(0,3,0,3)$	7	$(7,5,7,5)$	12	$(7,4,7,4)$	17	$(3,5,3,5)$
3	$(0,7,0,7)$	8	$(7,3,7,3)$	13	$(7,1,7,1)$	18	$(7,2,7,2)$
4	$(1,7,1,7)$	9	$(6,7,6,7)$	14	$(6,3,6,3)$	19	$(6,5,6,5)$
5	$(3,7,3,7)$	10	$(5,7,5,7)$	15	$(4,7,4,7)$	20	$(5,3,5,3)$

[3]We are not referring to v2 of LED.

3.9.2 SubCells (SC)

This operation substitutes the current value of each 4-bit cell with another 4-bit value, using the s-box of Present [9] given in Table 5.

Table 5: The s-box S of Present used in LED.

x	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$S[x]$	c	5	6	b	9	0	a	d	3	e	f	8	4	7	1	2

3.9.3 ShiftRows (SR)

This operation rotates the second, third and fourth row of the state matrix by one, two and three cells to the left.

3.9.4 MixColumnsSerial (MCS)

This operation multiplies the current value of the state with the following 4×4 matrix.

$$\begin{bmatrix} \mathtt{4} & \mathtt{1} & \mathtt{2} & \mathtt{2} \\ \mathtt{8} & \mathtt{6} & \mathtt{5} & \mathtt{6} \\ \mathtt{b} & \mathtt{e} & \mathtt{a} & \mathtt{9} \\ \mathtt{2} & \mathtt{2} & \mathtt{f} & \mathtt{b} \end{bmatrix}.$$

The result of the multiplication is the new value of the state.

4 Design Rationale

This section outlines the rationale for the design choices made for CiliPadi.

4.1 Key Lengths

We recommend two different key lengths, i.e. 128 and 256 bits. The former is meant for applications where resources such as area are very limited. The latter is proposed for applications where performance can be slightly sacrificed to gain more security.

4.2 Sponge

Our construction is based on the MonkeyDuplex [5] Sponge, which evolves from the original Sponge proposed in 2007 [7]. The versatile construction has been extensively scrutinized and deployed in numerous hash functions and AE proposals. These include Keccak [4] (standardized in SHA-3 and ISO/IEC 10118-3), PHOTON [20] (ISO/IEC 29192-5) and one of CAESAR's[4] final portfolio Ascon [16]. In particular, our use of different numbers of rounds in the initialization, message processing and finalization phases resembles that of Ascon and FIDES [8].

4.3 Permutation

The permutation function makes use of an unkeyed 2-round of the lightweight block cipher LED [21] as the F-function in a Type-II generalized feistel network (GFN) [32]. This is similar to the Simpira v2 permutation framework introduced by Gueron and Mouha [19]. To maximize diffusion, Simpira v2 utilizes a shuffling introduced by Suzaki and Minematsu [28], instead of the traditional left or right rotation of the input sub-blocks. The optimized shuffling allows us to achieve faster full diffusion of the input sub-blocks compared to the conventional Type-II GFN. A full diffusion means that all output sub-blocks are affected by all input sub-blocks.

Note that Simpira v2 was originally designed to utilize native AES instructions such as Intel's AES-NI present in many modern processors. The aim is to achieve a high throughput implementation. As there is no hardware-specific instructions for LED, this is not our ultimate aim. We chose to follow Simpira v2 due to its flexibility in extending to larger input lengths and also because it is easy to analyze its security with respect to differential and linear cryptanalysis.

By employing a Type-II GFN, it is trivial to extend the input lengths in multiple of 128 bits. The use of 2-round LED as the F-function allows us to borrow the security analysis done on Simpira v2, which utililizes 2-round AES instead.

The LED block cipher is chosen due to its lightweight construction and its similarity to the AES. In contrast to a 1-round LED that has a minimum of one active s-box, a 2-round LED has a minimum of 5 active s-boxes, which is identical to the AES [14]. This allows us to easily extend the results of Gueron and Mouha [19], whom originally use AES in the Simpira v2 framework, to CiliPadi.

The number of rounds for P_n^b, i.e. $b = 16$, is one round extra than the suggested number of rounds for Simpira v2 (i.e. 15) for $d = 4$ and $d = 6$. Furthermore, P_n^a is two rounds more than P_n^b, which should provide ample protection against tag forgery in the finalization phase.

5 Performance Analysis

We implemented and simulated CiliPadi-Mild using VHDL on Xilinx ISE and synthesized on xc6vlx760-2ff1760 FPGA chip inside the Virtex 6 development board. We use RTL approach and make use of an iterative type design. The other flavours of CiliPadi are obtained from our estimate based on the implementation of Mild. Table 6 shows a comparison of our implementation with other similarly designed AEs, i.e. Beetle [11] and ASCON [16]. These schemes are chosen due to the use of a sponge-based construction. Furthermore, the latter uses different number of rounds for the initialization and message encryption phases, which is similar to our design. The results for ASCON are obtained from Athena's database [17].

[4]Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR)

Table 6: Comparison of FPGA implementation of CiliPadi against other similar designs on Virtex 6. Results for Beetle and Ascon are taken from existing literature.

Scheme	LUTs	Slices	Freq. (MHz)	Gbps	Mbps/ LUT	Mbps/ Slice
Beetle[Light+]	616	252	381.592	1.879	3.050	7.369
Beetle[Secure+]	998	434	256	2.520	2.525	5.806
Ascon-128	1274	451	341.1	3.118	2.447	6.914
Ascon-128a	1587	547	358.6	5.099	3.213	9.322
CiliPadi-Mild	1052	303	639.959	1.138	1.082	3.756
CiliPadi-Medium	747	363	620	1.353	1.8112	3.727
CiliPadi-Hot	1268	373	631	1.685	1.329	4.517
CiliPadi-ExtraHot	1332	392	605	1.760	1.321	4.490

Beetle[Light+] and Beetle[Secure+] respectively use a 144-bit key (with 64 bits of security) and 256-bit key (with 121-bit security). On the other hand, both variants of Ascon employ a 128-bit key. The main differences between Ascon-128 and Ascon-128a are the message block size and numbers of permutation rounds. The former accepts a 64-bit block size while the latter, 128 bits.

The execution of CiliPadi-Mild and CiliPadi-Hot producing one ciphertext block and tag is 72 clock cycles. For CiliPadi-Medium and CiliPadi-ExtraHot, the number of clock cycles is 88. The primary member of CiliPadi, i.e. Mild, occupies 303 slices, which is about 20% higher than Beetle[Light+] but lower than other AEs compared in Table 6. However, our scheme provides 128-bit security as compared to Beetle[Light+]'s 64-bit. In the 256-bit key space, the two flavours of CiliPadi, i.e. Hot and ExtraHot consume fewer numbers of slices compared to Beetle[Secure+]. They even surpassed both variants of the 128-bit key Ascon. The numbers of slices for both Hot and ExtraHot flavours of CiilPadi are below 400 whereas the other compared AEs exceed this number. We believe that the implementation of CiliPadi can be further improved.

6 Security Analysis

6.1 Differential Cryptanalysis

In this section, we analyze the security of CiliPadi against differential cryptanalysis both theoretically and experimentally. We then extend some of these findings to linear cryptanalysis. Note that although there is no LED round subkeys to recover, such analysis is still useful since our analysis falls in the known-key attack model [24]. In our case, the known-key is the round constants.

6.1.1 Preliminaries

P is based on the Simpira v2 framework which is a d-line Type-II GFN. Instead of using AES like Simpira, CiliPadi uses an unkeyed 2-round LED as its round function, F. The design of LED shares same diffusion properties as AES, thus 2 rounds of LED has 5 active s-boxes (AS) [14]. In other words, F has a minimum of 5 active s-boxes given a non-zero input. The maximum differential probability of its s-box (which is essentially PRESENT's s-box [9]) is 2^{-2}. We evaluate

CiliPadi using two approaches, the first of which is based on the notions of P as a random permutation and the second is based on identifying collision-producing differentials. An upper bound of the differential probability for both approaches will be defined based on the number of "active F-functions". We first provide some brief definitions for these concepts:

P as a random permutation: It has been shown that the security of sponge and duplex constructions rely on their underlying permutations being random [6, 5]. To evaluate P as a random permutation, we take into consideration the entire internal state, S as a whole. The maximum differential probability for P must be 2^{-256} and 2^{-384} for internal states of $n = 256$ and $n = 384$ respectively, to demonstrate some semblance to a random permutation. To obtain a differential path for the entire state S, a related-key/related nonce differential attack can be applied on the initialization phase, whereby an adversary is allowed to inject differences in either the key or nonce (or both of them). We can then define a differential path as $\Delta X \xrightarrow{\hat{p}} \Delta Y$, where $\hat{p}$ is probability that an input difference ΔX leads to an output difference ΔY. We refer to $\hat{p}$ as the differential probability. ΔX and ΔY are defined as

$$\Delta X = (K_1 \oplus K_2) \| (N_1 \oplus N_2),$$

and

$$\Delta Y = P_n^a(K_1 \| N_1) \oplus P_n^a(K_2 \| N_2),$$

respectively.

Thus, if $\Delta X \xrightarrow{\hat{p}} \Delta Y$ holds with $\hat{p} > 2^{-n}$, P is susceptible to a trivial distinguishing attack:

i. Initialize a counter, $c = 0$.

ii. Generate 2^n related-key/related-nonce pairs corresponding to ΔX.

iii. Encrypt each key-nonce pair to obtain the corresponding output difference.

iv. For each key-nonce pair that fulfills, $\Delta X \rightarrow \Delta Y$, increment c.

v. Statistically, the distinguishing attack is successful if $c \approx 2^{\hat{p}}$.

Note that computing the differential path requires that random subkeys be used for each round of the underlying `LED` cipher to make the s-box inputs independent. These subkeys can be simulated by the addition of round constants.We also note that exhaustively searching through a state space of 256 or 384 bits exceeds today's computing capability. Therefore, these results are only of academic interest.

Collision-producing differential: In the AD authentication and message encryption phases, a straightforward application of differential cryptanalysis is difficult because the initial capacity state S_c is unknown to an adversary. However, we can investigate collision-producing differentials for CiliPadi which are differentials that have differences in S_r for both the input and outputs of P, but have zero differences for S_c. Such differentials may be useful in forgery attacks. A collision-producing differential can be defined as

$$\Delta X \| 0_2^c \xrightarrow{\hat{p}} \Delta Y \| 0_2^c, \tag{1}$$

where ΔX can be introduced by injecting a difference in the message block, $\Delta Y \| 0^c = P_n^b(\Delta X \| 0_2^c)$, and $\hat{p}$ is the probability that the differential holds.

Active F-function analysis: To theoretically estimate the upper bounds of the differential probability, we use the notion of "active F-functions" (AF). An F-function is considered to be active when it receives a non-zero input, similar to the concept of AS. We have implemented a searching algorithm to compute the number of AF for each round which is equivalent to identifying the number of AS for a regular GFN. The algorithm is based on a modified version of Matsui's branch-and-bound search proposed in [12].

To simplify explanations, we will use the concept of truncated differentials, whereby every 64 bits of the concrete difference is represented as 1 bit in the truncated difference. A non-zero 64-bit block results in a non-zero truncated bit, and vice versa. E.g. for $d = 4$, if a concrete difference consists of four 64-bit differences, $\Delta X = (0_2^{63} \| 1_2) \| (0_2^{63} \| 1_2) \| (0_2^{64}) \| (0_2^{64})$, the corresponding truncated difference is $\Delta X^T = (1100)$. Thus, the maximum Hamming weight of the truncated difference is equal to d. The odd numbered bits (1,3,6) of the truncated difference will pass through the F-function, thus "activating" it.

Using the searching algorithm, we identify the number of AF for $d = 4$ and $d = 6$ for up to 30 rounds as shown in Tables 7 and 8 respectively. In the following sections, we use this methodology to first evaluate P as a random permutation before examining its security against the distinguishing attack.

Table 7: Active F-function distribution for CiliPadi-Mild/Medium $(d = 4)$.

Round	1	2	3	4	5	6	7	8	9	10
AF	0	1	2	3	4	6	6	8	10	11
Round	11	12	13	14	15	16	17	18	19	20
AF	12	12	12	13	14	15	16	18	18	19
Round	21	22	23	24	25	26	27	28	29	30
AF	20	21	22	24	24	25	26	27	28	30

Table 8: Active F-function distribution for CiliPadi-Hot/ExtraHot $(d = 6)$.

Round	1	2	3	4	5	6	7	8	9	10
AF	0	1	2	3	4	6	8	10	11	12
Round	11	12	13	14	15	16	17	18	19	20
AF	13	14	15	17	19	21	22	23	24	25
Round	21	22	23	24	25	26	27	28	29	30
AF	26	28	30	32	33	34	35	36	37	39

6.1.2 P as a Random Permutation

For P to approximate a random permutation, the differential probability should be at most 2^{-256} (2^{-384}) for $d = 4$ $(d = 6)$. As the differential probability of the s-box is 2^{-2}, this requires at least $\frac{256}{2} = 128$ $(\frac{384}{2} = 192)$ AS. A conservative approach is to assume each AF to contain only 5

AS as was the assumption made for a Type-II GFN that has 2 substitution layers interleaved with a single maximum distance separable (MDS)-based diffusion layer [10] (same as our 2-round LED with the exception that ours have an extra diffusion layer). Based on this approach, therefore, $\frac{256}{2\times 5} \approx 26$ ($\frac{384}{2\times 5} \approx 39$) AF is required in order for P to resist differential cryptanalysis. Indeed, 26 (39) AF gives $26 \times 5 = 130$ ($39 \times 5 = 195$) AS. Based on this rough estimate, according to Table 7 (8), P needs to have at least 27 (30) rounds for $d = 4$ ($d = 6$) to avoid any biases from a random permutation.

However, since P makes use of LED, which inherits the wide trails strategy of the AES, we can improve the previous analysis. As illustrated in Figure 6 and proven by the designers of AES, any 4-round differential path provides a minimum of 25 active s-boxes. Suppose that a particular round in LED has one AF where the internal differential paths looks like the first 2 rounds of the 4-round path depicted in the figure. We can observe that MCS causes all 4-bit cells to have nonzero output difference. Then, these 16 nonzero differences become the input to the next subsequent F-function which activate 16 AS in the first LED round and another 4 AS in the second LED round. It is therefore not possible for this second AF to have 5 AS as assumed before. Due to the wide trail strategy, this second AF is guaranteed to contain 20 AS. The AS pattern is $1 \to 4 \to 16 \to 4 \to 1$.

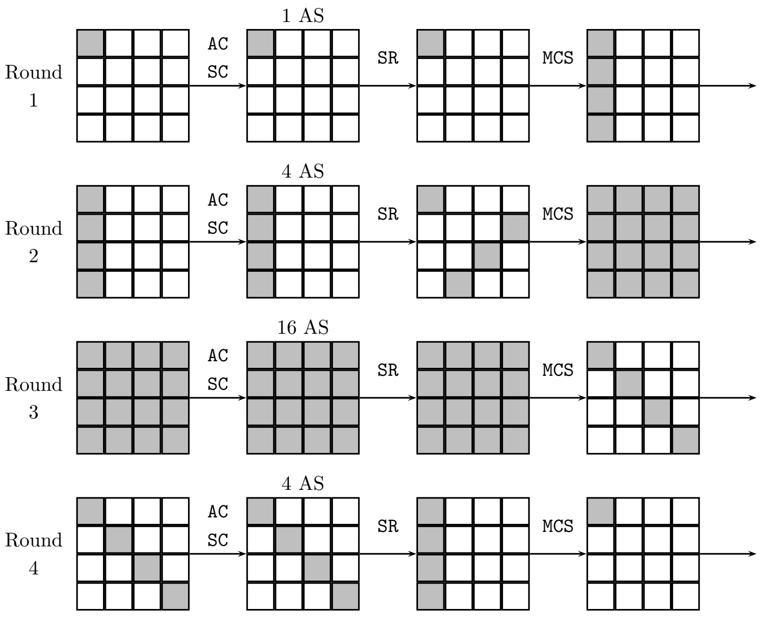

Figure 6: A 4-round differential path for LED that guarantees at least 25 active s-boxes (AS)

Table 9 expands the AF distribution given in Tables 7 and 8 by showing examples of truncated differential paths for P_n^a for all flavours of CiliPadi. As given earlier in this section, for $d = 4$, 128 AS are required in order for P to be resistant to differential cryptanalysis. For $d = 6$, the number of AS is 192. Based on Table 9, the truncated differential path for P_{256}^{18} and P_{256}^{20} each contains a minimum of 180 and 185 AS, respectively. On the other hand, the path for P_{384}^{18} and P_{384}^{20} each comprises 265 and 290 AS, respectively. These numbers for CiliPadi are beyond the required number of AS. The upper bounds of the differential probability $\hat{p}_u$ of P_n^a for all variants of CiliPadi are shown in Table 10. For each variant, we also provide the number of truncated paths that correspond to the number of AF. Note that a 6-round iterative truncated differential with 6 AF ($0001 \to 0001$), and a 16-round iterative truncated differential with 22 AF ($000001 \to 000001$) exists for $d = 4$ and $d = 6$ respectively.

Table 9: Numbers of AF and AS for truncated differential paths for P_n^a.

	Mild (P_{256}^{18})			Medium (P_{256}^{20})			Hot (P_{384}^{18})			ExtraHot (P_{384}^{20})		
Rnd.	ΔX	AF	AS	ΔX	AF	AS	ΔX	AF	AS	ΔX	AF	AS
1	0001	0	0	0001	0	0	000001	0	0	000011	1	5
2	0010	1	5	0010	1	5	001000	1	5	000001	0	0
3	0110	1	20	0110	1	20	010010	1	20	001000	1	5
4	1110	2	10	1110	2	10	101001	2	10	010010	1	20
5	0111	1	20	0111	1	20	111110	3	60	101001	2	10
6	1100	1	5	1100	1	5	011111	2	10	111110	3	60
7	0001	0	0	0001	0	0	110001	1	20	011111	2	10
8	0010	1	5	0010	1	5	001100	1	5	110001	1	20
9	0110	1	20	0110	1	20	010000	0	0	001100	1	5
10	1110	2	10	1110	2	10	100000	1	5	010000	0	0
11	0111	1	20	0111	1	20	100100	1	20	100000	1	5
12	1100	1	5	1100	1	5	100110	2	10	100100	1	20
13	0001	0	0	0001	0	0	101111	3	60	100110	2	10
14	0010	1	5	0010	1	5	110111	2	10	101111	3	60
15	0110	1	20	0110	1	20	000111	1	20	110111	2	10
16	1110	2	10	1110	2	10	000011	1	5	000111	1	20
17	0111	1	20	0111	1	20	000001	0	0	000011	1	5
18	1100	1	5	1100	1	5	001000	1	5	000001	0	0
19				0001	0	0				001000	1	5
20				0010	1	5				010010	1	20

Table 10: Differential probability upper bounds for P_n^a

CiliPadi-	n	a	AF	AS	$\hat{p}_u$	Truncated Paths
Mild	256	18	18	180	2^{-360}	122
Medium	256	20	19	185	2^{-370}	10
Hot	384	18	23	265	2^{-530}	144
ExtraHot	384	20	25	290	2^{-580}	36

Practical Confirmation: We now experimentally confirm that our "active F-function" estimation is a conservative lower bound, and that the actual number of AS per AF would be higher than 5 as the number of rounds increases. In other words, the number of AF provides us with an upper bound in terms of differential probability. To perform the differential search, we leverage upon the methodology described in [13]. Here, we focus on only the CiliPadi-Mild as a proof-of-concept and use the truncated differential path shown in Table 9 as a guide. We limit our input difference to have a hamming weight of 1 (only 1 bit out of any 64-bit word will be active at one time). Due to computational limitations, we bound the search based on each round of LED as:

- **LED Round 1:** Based on the input difference, if the number of activated s-boxes is more than 8, we limit the number of branches to 2 for each s-box, whereby we select the two branches with the highest differential probability. Otherwise, we search all branches.
- **LED Round 2:** Based on the input difference, if the number of activated s-boxes is more than 4, we limit the number of branches to 1 for each s-box, whereby we select the branch with the highest differential probability. Otherwise, we search all branches.

Based on this methodology, we found a 4-round concrete path following the truncated differential path $0001 \to 0111$ with a probability of 2^{-140}:

$$\Delta X = 0_2^{64} \| 0_2^{64} \| 0_2^{64} \| (0_2^{28} \| 2 \| 0_2^{32}),$$

$$\Delta Y = 0_2^{64} \| \texttt{7f46a6de679866ce} \| \texttt{8f01218a4117896f} \| (0_2^{28} \| 2 \| 0_2^{32}),$$

where ΔY is post-shuffle. The breakdown of the concrete differential path, the number of AS per round, and the comparison to our lower bound AS_l, is shown in Table 11. In the second round, there are 5 AF which leads to a probability of $(2^{-2})^5 = 2^{-10}$, verifying the correctness of our implementation. However, although the third round has only 1 AF, the actual number of AS is 28 due to the strong diffusion capability of LED. It confirms our claim that the number of "active F-functions" can be used as a conservative estimate of the security margin, and will lead to a conservative lower bound in terms of security margin (and equivalently, an upper-bound in terms of differential probability).

Table 11: Example of a concrete differential path for P_{256}^a.

Concrete Differential				AS	AS$_l$
0_2^{64}	0_2^{64}	0_2^{64}	$(0_2^{28}\|2\|0_2^{32})$	0	0
0_2^{64}	0_2^{64}	$(0_2^{28}\|2\|0_2^{32})$	0_2^{64}	5	5
0_2^{64}	$(0_2^{28}\|2\|0_2^{32})$	`c25adcad9fdb44b1`	0_2^{64}	28	20
$(0_2^{28}\|2\|0_2^{32})$	`c25adcad9fdb44b1`	`7f46a6de679866ce`	0_2^{64}	36	10
0_2^{64}	`7f46a6de679866ce`	`8f01218a4117896f`	$(0_2^{28}\|2\|0_2^{32})$	-	-

6.1.3 Collision-Producing Differentials of CiliPadi

The number of AF for a collision-producing truncated differential for CiliPadi-Mild and CiliPadi-ExtraHot can be identified by fixing both the input and output truncated differences to "1000" and "110000" respectively (i.e. $1000 \to 1000$ and $110000 \to 110000$ because r is a multiple of 64. For ease of analysis, we use $1000 \to 1000$ as the truncated differential for CiliPadi-Medium, by setting the remaining $96 - 64 = 32$ bits of the bitrate part to nonzero. For CiliPadi-Hot, we use $100000 \to 100000$. We then employ the same searching algorithm to identify the truncated differential path with the lowest number of AF for P_n^b. The results are summarized in Table 12 where $\hat{p}_{col}$ denotes the probability of the collision-inducing path. The truncated paths corresponding to each of CiliPadi's variants are as shown in Table 13. Note that a 6-round iterative truncated collision-producing differential exists for $d = 4$, where $1000 \to 1000$ with 6 AF.

Table 12: Collision-producing differential probability upper bounds for P_n^b.

CiliPadi-	n	b	AF	AS	$\hat{p}_{col}$
Mild	256	16	18	180	2^{-360}
Medium	256	18	18	180	2^{-360}
Hot	384	16	22	260	2^{-520}
ExtraHot	384	18	26	310	2^{-620}

Table 13: Numbers of AF and AS for truncated collusion-producing differential paths for P_n^b.

	Mild (P_{256}^{16})			Medium (P_{256}^{18})			Hot (P_{384}^{16})			ExtraHot (P_{384}^{18})		
Rnd.	ΔX	AF	AS	ΔX	AF	AS	ΔX	AF	AS	ΔX	AF	AS
1	1000	1	5	1000	1	5	100000	1	5	110000	1	5
2	1001	1	20	1001	1	20	100100	1	20	000100	0	0
3	1011	2	10	1011	1	10	100110	2	10	000010	1	5
4	1101	1	20	1101	2	20	101111	3	60	001001	1	20
5	0011	1	5	0011	1	5	110111	2	10	011010	2	10
6	0100	0	0	0100	0	0	000111	1	20	111011	3	60
7	1000	1	5	1000	1	5	000011	1	5	010111	1	5
8	1001	1	20	1001	1	5	000001	0	0	101011	3	60
9	1011	2	10	1011	2	10	001000	1	5	110111	2	10
10	1101	1	20	1101	2	5	010000	1	20	000111	1	20
11	1011	2	10	0011	1	5	100000	2	10	000011	1	5
12	1101	1	20	0100	0	0	100100	3	60	000001	0	0
13	1011	2	10	1000	1	5	100110	2	10	001000	1	5
14	1101	1	20	1001	1	20	101111	1	20	010010	1	20
15	0011	1	5	1011	2	10	110101	1	5	101001	2	10
16	0100	0	0	1101	2	20	001110	0	0	111110	3	45
17				0011	1	5				111101	2	10
18				0100	0	0				011100	1	20

Practical Confirmation: We now experimentally confirm that the collision probabilities in Table 12 provides conservative lower bounds. Again, we target CiliPadi-Mild as a proof-of-concept and use the truncated differential path shown in Table 13 as a guide. Based on the same methodology described in Section 6.1.2, we found a 3-round concrete path following the truncated differential path with a probability of 2^{-140}:

$$\Delta X = (0_2^{28} \| 2 \| 0_2^{32}) \| 0_2^{64} \| 0_2^{64} \| 0_2^{64},$$

$$\Delta Y = \texttt{8f01218a4117896f} \| (0_2^{28} \| 2 \| 0_2^{32}) \| 0_2^{64} \| \texttt{7f46a6de679866ce},$$

where ΔY is post-shuffle. The above differential path contains a total of 69 AS, which is significantly higher than the theoretical lower bound of 35 AS for a 3-round differential, as shown in Table 13.

6.1.4 Practical Security Bounds

In practice, the best cryptanalytic attack requires less computational complexity than an exhaustive search of the secret key. CiliPadi has key sizes of 128 and 256 bits, thus any statistical distinguisher for a successful attack must have a probability higher than 2^{-128} and 2^{-256} respectively. Based on Tables 10 and 12, the theoretical upper bounds of the differential probability indicate that all flavours of CiliPadi are highly resistant to differential cryptanalysis and collision attacks. In reality, the differential probabilities are much lower as depicted in the practical confirmation experiments. Therefore, CiliPadi is expected to thwart any differential type attacks.

6.2 Full Bit Diffusion

With the availability of the full AF distribution, we can determine the minimum number of rounds for P to achieve full bit diffusion. Here, the findings from Simpira v2 are directly applicable because the underlying round functions for both Simpira and CiliPadi have the same diffusion properties. For $d = 4$, full bit diffusion is achieved after $4d - 6 = 16 - 6 = 10$ F-functions [19]. Based on Table 7, 9 rounds of P is sufficient for full bit diffusion. As for $d = 6$, full bit diffusion is achieved after 5 rounds [28]. Thus, the current number of rounds of P for all variants of CiliPadi are sufficient to achieve full bit diffusion.

6.3 Extension to Linear Cryptanalysis

The previous findings on differential cryptanalysis can be trivially extended to linear cryptanalysis due to the duality between linear and differential cryptanalysis [25, 10], and also due to PRESENT's s-box having a linear probability of 2^{-2}. Thus, all the results in the previous subsections are applicable to linear cryptanalysis.

7 Strengths and Weaknesses

The following list the expected strengths and weaknesses of CiliPadi.

7.1 Strengths

CiliPadi has the following advantages:

- It is trivial to expand the length of the permutation in multiple of 128 bits due to the use of a Type-II GFN.
- The bitrate can be adjusted to allow different plaintext and tag lengths.
- The design is based on the sponge construction, which have been extensively analyzed and employed in SHA-3 and one of the CAESAR portfolio Ascon.
- Any AES-like block cipher or permutation can be adopted in the F-function to replace the LED block cipher, if desired.

7.2 Weaknesses

The known limitations of CiliPadi are:

- The processing of the message and ciphertext blocks cannot be parallelized because due to the sequential processing of the input blocks.

- The permutation can only be expanded in multiple of 128 bits. Extending with a smaller granularity, e.g. 32 bits, is not supported. This can be addressed by using a smaller block cipher as the F-function such as KATAN and KTANTAN [15].

Acknowledgement The authors thank Je Sen Teh and Yasir Amer Abbas for providing additional security analysis and implementation results.

Conflicts of Interest The authors declare no conflict of interest in the publication of this work.

References

[1] Bagheri, N (2019). *OFFICIAL COMMENT: CiliPadi. Official comments received on CiliPadi*. NIST Lightweight Cryptography Project, Gaithersburg, MD. https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-1/official-comments/CiliPadi-official-comment.pdf.

[2] M. Bellare, T. Kohno & C. Namprempre (2004). Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the encode-then-encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, *7*(2), 206–241.

[3] M. Bellare & C. Namprempre (2008). Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, *21*(4), 469–491.

[4] G. Bertoni, J. Daemen, M. Peeters & G. V. Assche (2011). *The Keccak SHA-3 Submission, Version 3*. SHA-3 Cryptographic Hash Algorithm Competition, Gaithersburg, MD. http://keccak.noekeon.org.

[5] G. Bertoni, J. Daemen, M. Peeters & G. V. Assche (2012). *Permutation-Based Encryption, Authentication and Authenticated Encryption*. DIAC – Directions in Authenticated Ciphers, Stockholm, Sweden.

[6] G. Bertoni, J. Daemen, M. Peeters & G. Van Assche (2012). Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Selected Areas in Cryptography*, pp. 320–337. Springer, Berlin, Heidelberg.

[7] G. Bertoni, J. Daemen, M. Peeters & G. V. (2007) (2007). *Sponge Functions*. ECRYPT Workshop on Cryptographic Hash Function, Barcelona, Spain.

[8] B. Bilgin, A. Bogdanov, M. Knežević, F. Mendel & Q. Wang (2013). FIDES: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In *Cryptographic Hardware and Embedded Systems*, pp. 142–158. Springer, Berlin, Heidelberg.

[9] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin & C. Vikkelsoe (2007). PRESENT: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems*, pp. 450–466. Springer, Berlin, Heidelberg.

[10] A. Bogdanov & K. Shibutani (2013). Generalized feistel networks revisited. *Designs, Codes and Cryptography*, *66*, 75–97.

[11] A. Chakraborti, N. Datta, M. Nandi & K. Yasuda (2018). Beetle family of lightweight and secure authenticated encryption ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, *2018*(2), 218–241.

[12] J. Chen, A. Miyaji, C. Su & J. S. Teh (2016). Accurate estimation of the full differential distribution for general feistel structures. In *Information Security and Cryptology*, pp. 108–124. Springer, Cham.

[13] J. Chen, J. Teh, Z. Liu, C. Su, A. Samsudin & Y. Xiang (2017). Towards accurate statistical analysis of security margins: New searching strategies for differential attacks. *IEEE Transactions on Computers*, *66*(10), 1763–1777.

[14] J. Daemen & V. Rijmen (2002). *The Design of Rijndael, AES – The Advanced Encryption Standard*. Springer-Verlag, Berlin, Heidelberg.

[15] C. De Cannière, O. Dunkelman & M. Knežević (2009). KATAN and KTANTAN – a family of small and efficient hardware-oriented block ciphers. In *Cryptographic Hardware and Embedded Systems*, pp. 272–288. Springer, Berlin-Heidelberg.

[16] C. Dobraunig, M. Eichlseder, F. Mendel & M. Schläffer (2016). *Ascon v1.2: Submission to the CAESAR Competition*. CAESAR Competition, Nagoya, Japan. https://competitions.cr.yp.to/round3/asconv12.pdf.

[17] George Mason University (2014). *Authenticated Encryption FPGA Ranking*. ATHENa: Automated Tools for Hardware EvaluatioN, Fairfax County, Virginia. https://cryptography.gmu.edu/athenadb/fpga_auth_cipher/table_view.

[18] V. D. Gligor & P. Donescu (2002). Fast encryption and authentication: XCBC encryption and XECB authentication modes. In *Fast Software Encryption*, pp. 92–108. Springer, Berlin, Heidelberg.

[19] S. Gueron & N. Mouha (2016). Simpira v2: A family of efficient permutations using the AES round function. In *Advances in Cryptology*, pp. 95–125. Springer, Berlin, Heidelberg.

[20] J. Guo, T. Peyrin & A. Poschmann (2011). The PHOTON family of lightweight hash functions. In *Advances in Cryptology*, pp. 222–239. Springer, Berlin, Heidelberg.

[21] J. Guo, T. Peyrin, A. Poschmann & M. Robshaw (2011). The LED block cipher. In *Cryptographic Hardware and Embedded Systems*, pp. 326–341. Springer, Berlin, Heidelberg.

[22] C. S. Jutla (2001). Encryption modes with almost free message integrity. In *Advances in Cryptology*, pp. 529–544. Springer, Berlin, Heidelberg.

[23] J. Katz & M. Yung (2001). Unforgeable encryption and chosen ciphertext secure modes of operation. In *Fast Software Encryption*, pp. 284–299. Springer, Berlin, Heidelberg.

[24] L. R. Knudsen & V. Rijmen (2007). Known-key distinguishers for some block ciphers. In *Advances in Cryptology*, pp. 315–324. Springer, Berlin, Heidelberg.

[25] M. Matsui (1995). On correlation between the order of s-boxes and the strength of DES. In *Advances in Cryptology*, pp. 366–375. Springer, Berlin-Heidelberg.

[26] K. G. Paterson & G. J. Watson (2012). Authenticated-encryption with padding: A formal security treatment. In *Cryptography and Security: From Theory to Applications*, pp. 83–107. Springer, Berlin, Heidelberg.

[27] P. Rogaway, M. Bellare, J. Black & T. Krovetz (2001). Ocb: A block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security*, pp. 196–205. ACM, Philadelphia, USA.

[28] T. Suzaki & K. Minematsu (2010). Improving the generalized feistel. In *Fast Software Encryption*, pp. 19–39. Springer, Berlin, Heidelberg.

[29] M. S. Turan, K. A. McKay, c. Çalık & L. Bassham (2019). *Status Report on the First Round of the NIST Lightweight Cryptography Standardization Process*. National Institute of Standards and Technology, NISTIR 8268, Gaithersburg, MD. https://doi.org/10.6028/NIST.IR.8268.

[30] S. Vaudenay (2002). Security flaws induced by CBC padding – applications to SSL, IPSEC, WTLS ... In *Advances in Cryptology – EUROCRYPT 2002*, pp. 534–545. Springer-Verlag, Berlin, Heidelberg.

[31] M. R. Z'aba, N. Jamil, M. S. Rohmad, H. Abdul Rani & S. Shamsuddin (2019). *The Cili-Padi Family of Lightweight Authenticated Encryption*. NIST Lightweight Cryptography Project, Gaithersburg, MD.

[32] Y. Zheng, T. Matsumoto & H. Imai (1990). On the construction of block ciphers provably secure and not relying on any unproved hypotheses. In *Advances in Cryptology*, pp. 461–480. Springer, New York, NY.

8 Appendices

The following are the test vectors for the recommended flavours of CiliPadi.

Table 14: CiliPadi test vectors.

128-bit Key	
Key	`000102030405060708090a0b0c0d0e0f`
Nonce	`000102030405060708090a0b0c0d0e0f`
AD	`000102030405060708090a0b0c0d0e0f`
Plaintext	`000102030405060708090a0b0c0d0e0f`
CiliPadi-Mild	
Ciphertext	`3fb9e70d3702c712d407f60f617e43d3`
Tag	`68b96217dd237301`
CiliPadi-Medium	
Ciphertext	`e121ea7f97d9ba3c93f8e0fe7bd3b247`
Tag	`d8c081563c28aa673557beac`
256-bit Key	
Key	`000102030405060708090a0b0c0d0e0f`
	`101112131415161718191a1b1c1d1e1f`
Nonce	`000102030405060708090a0b0c0d0e0f`
AD	`000102030405060708090a0b0c0d0e0f`
	`101112131415161718191a1b1c1d1e1f`
Plaintext	`000102030405060708090a0b0c0d0e0f`
	`101112131415161718191a1b1c1d1e1f`
CiliPadi-Hot	
Ciphertext	`5dca237e5d333998aaabafa37faeea11`
	`6763b68822e11d8370a073182677eadd`
Tag	`8002b646e2d8a551ea64c40b`
CiliPadi-ExtraHot	
Ciphertext	`54c5f1b5ca2e440f9db6fc4c0dfe2c72`
	`57371d685219fbf19a3ea3f6aedc81c3`
Tag	`aa00ba04115fdf838242b021e80de375`