# Yarará and Coral v1

Miguel Montes
UNDEF CRUC-IUA
mmontes@iua.edu.ar

Daniel Penazzi
Famaf - Universidad Nacional de Córdoba -CIEM
penazzi@famaf.unc.edu.ar

March 29, 2019

## 1 Introduction

We present here two related lightweight cryptographic algorithsms, Yarará and Coral.

Yarará is an authenticated encryption algorithm with a key length of 128 bits, a nonce length of 128 bits, and a tag length of 128 bits. Coral is a 256-bit hash function. For both algorithms the input and the output are byte strings, that is, it is assumed that the lengths of the associated data, plaintext, ciphertext are all multiples of 8 bits.

Both are permutation based, and share the same underlying permutation.

They are based on the sponge mode [6] and the duplex sponge mode [2].

Both algorithms have a state $S$ of size $b = 256$ bits. We can see the state as four 64-bit words, $x_0$, $x_1$, $x_2$ and $x_3$. We denote the part of the state dedicated to absorbing as $S_r$ and the remaining part as $S_c$.

Yarará has a rate of $r = 64$ bits and a capacity of $c = 256 - r = 192$ bits. $S_r$ corresponds to the first word of the state, that is, $x_0$, as shown in figure 1.

Coral has a rate of $r = 32$ bits and a capacity of $c = 256 - r = 224$ bits. The 32 bits of $S_r$ are the 32 most significant bits of $x_0$.

| $S_r$ | $S_c$ | | |
|---|---|---|---|
| $x_0$ | $x_1$ | $x_2$ | $x_3$ |

Figure 1: Yarará: 256-bit state

We describe Yarará in section 2, Coral in section 3 and the shared permutation in section 4.

### 1.1 Notation

We use the following notation:

| | |
|---|---|
| $0^n$ | A bitstring of $n$ zeros |
| $S$ | The 256-bit state. |
| $r$ | The rate. |
| $c$ | The capacity. |
| $S_r$ | The r-bit rate part of the state, where input is absorbed and output is produced. |
| $S_c$ | The c-bit capacity of the state. |
| $x_0, x_1, x_2, x_3$ | The four 64-bit words of the state. |
| $\pi_I$, $\pi_{AD}$, $\pi_{AE}$, $\pi_F$ | The permutations. |
| $A$, $AD$ | Associated data. |
| $C$, $P$ | Ciphertext and plaintext. |
| $M$ | Message. |
| $H$ | The 256-bit hash result. |
| $K$, $N$, $T$ | The 128-bit key, nonce and tag. |
| $P_i$, $C_i$, $T_i$, $M_i$, $H_i$ | $r$-bit chunk of $P$, $C$, $T$, $M_i$, $H_i$. In the case of the last chunk, the length could be less than $r$. |
| $MSB_n(X)$ | $n$ most significant (leftmost) bits of $X$. |
| $LSB_n(X)$ | $n$ least significant (rightmost) bits of $X$. |

We use the *big-endian* convention for dealing with 64-bit and 32-bit words, that is, the most significant bit is stored in the left-most position of each word.

When a byte string is loaded into a 64-bit or 32-bit word, the first byte of the string will be the most significant byte of the word, the second byte will be the second most significant byte, and so on.

# 2 The AEAD algorithm Yarará

## 2.1 Mode of operation

Yarará operates in four stages. Here we describe the encryption procedure.

First the 256-bit state is initialized with the 128-bit key and 128-bit nonce, and transformed via the initial permutation $\pi_I$ (all permutations are defined in section 4).

Then the associated data is padded and absorbed in $r$-bit chunks if and only if its length is not zero. After each absorption the state is transformed via the permutation $\pi_{AD}$. In either case, before the plaintext processing phase, a single domain separation bit is xored with the least significant bit of the state.

In the third phase the plaintext is processed. Each $r$-bit block of the padded plaintext is xored with the first $r$ bits of the state ($S_r$) to produce both a block of ciphertext and the new value of $S_r$, and then the permutation $\pi_{AE}$ is applied to the state. The last block of ciphertext is of the same length as the last unpadded block of plaintext.

Both the AD and the plaintext are padded with a single bit with value 1 and just enough zeros to make their lengths a multiple of $r$ bits. In practice we assume all inputs are byte strings, so the padding consists of the byte valued 0x80 and enough zero bytes to make the length multiple of $r/8$.

After the last block of input is processed the permutation $\pi_F$ is applied to the state the number of times needed for producing the tag, $r$-bits at a time. As we specify a tag length of 128 bits, and a rate $r$ of 64, we need two applications of $\pi_F$.

The decryption procedure differs in the third phase. Each block of ciphertext is xored with $S_r$ to produce the plaintext, and also becomes the new $S_r$.

The decryption procedure only releases the plaintext if the received tag is identical to the calculated tag. If they are different, it returns an error.

The encryption procedure is illustrated in figure 2 and specified in figure 4, while the decryption procedure is shown in figure 3 and specified in figure 5.

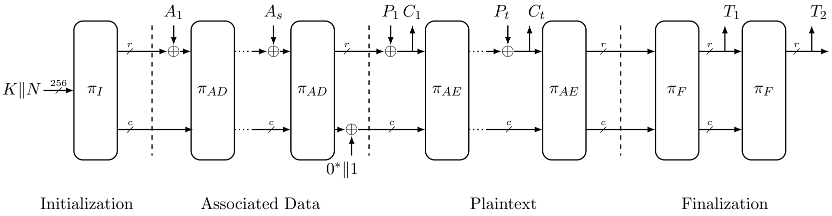

Figure 2: Encryption

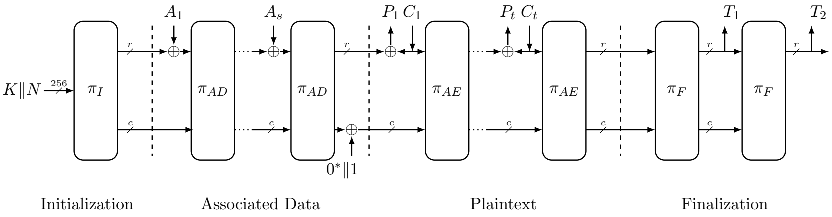

Figure 3: Decryption

```
AEAD-Encrypt$(K, N, A, P)$
// Initialization
$S = K \parallel N$
$S = \pi_I(S)$
// Processing Associated Data
if $|A| \neq 0$
    $s = \lfloor |A|/r \rfloor + 1$
    $A_1 \ldots A_s = \mathrm{PAD}(A)$
    for $i = 1$ to $s$
        $S_r = S_r \oplus A_i$
        $S = \pi_{AD}(S)$
$S = S \oplus (0^{255} \parallel 1)$
// Processing Plaintext
$t = \lfloor |P|/r \rfloor + 1$
$\ell = |P| \mod r$
$P_1 \ldots P_t = \mathrm{PAD}(P)$
for $i = 1$ to $t - 1$
    $S_r = S_r \oplus P_i$
    $C_i = S_r$
    $S = \pi_{AE}(S)$
$S_r = S_r \oplus P_t$
$C_t = MSB_\ell(S_r)$
$S = \pi_{AE}(S)$
// Finalization
$u = \lfloor |T|/r \rfloor$
$\tau = |T| \mod r$
for $i = 1$ to $u$
    $S = \pi_F(S)$
    $T_i = S_r$
if $\tau > 0$
    $S = \pi_F(S)$
    $T_{u+1} = MSB_\tau(S_r)$
return $C, T$
```

Figure 4: Encryption procedure

```
AEAD-Decrypt$(K, N, A, C, T)$
// Initialization
$S = K \parallel N$
$S = \pi_I(S)$
// Processing Associated Data
if $|A| \neq 0$
        $s = \lfloor |A|/r \rfloor + 1$
        $A_1 \ldots A_s = \mathrm{PAD}(A)$
        for $i = 1$ to $s$
                $S_r = S_r \oplus A_i$
                $S = \pi_{AD}(S)$
$S = S \oplus (0^{255} \parallel 1)$
// Processing Ciphertext
$t = \lceil |C|/r \rceil$
$\ell = |C| \mod r$
for $i = 1$ to $t - 1$
        $P_i = S_r \oplus C_i$
        $S_r = C_i$
        $S = \pi_{AE}(S)$
$P_t = MSB_\ell(S_r) \oplus C_t$
$S_r = C_t \parallel (LSB_{r-\ell}(S_r) \oplus (1 \parallel 0^{r-\ell-1}))$
$S = \pi_{AE}(S)$
// Finalization
$u = \lfloor |T|/r \rfloor$
$\tau = |T| \mod r$
for $i = 1$ to $u$
        $S = \pi_F(S)$
        $T'_i = S_r$
if $\tau > 0$
        $S = \pi_F(S)$
        $T'_{u+1} = MSB_\tau(S_r)$
if $T == T'$
        return $P$
else
        return $\perp$
```

Figure 5: Decryption procedure

## 2.2 Security Claims

We claim 128 bits of security regarding the confidentiality of the plaintext, and also for the integrity of the plaintext, associated data and public message number.

If the tag is truncated to $\tau$ bits, the security regarding integrity is reduced to $\tau$.

That is, we expect that any attack on the confidentiality of the plaintext will need $2^{128}$ effort and if the length of the tag $T$ is $\tau$, a forgery of the plaintext, associated data or public message number cannot be made with probability greater than $2^{-\tau}$ (i.e., an expected $2^{\tau}$ attempts need to be made before a forgery is accepted as valid). However, in accordance to the NIST call, we do not distinguish between messages one of which is a truncation of the other by a number of bits less than 8.

The public message number should be a nonce. The cipher does not promise any integrity or confidentiality if the legitimate key holder uses the same nonce to encrypt two different (plaintext, associated data) pairs under the same key.

These claims hold if the total number of processed blocks (both AD and plaintext) does not exceed $2^{64}$ blocks, that is, $2^{67}$ bytes.

## 2.3 Advantages and limitations

### 2.3.1 Advantages

- It uses well known and tested primitives.
- It allows efficient bit-sliced implementations in software. All operations are defined in term of 64-bit word operations such as MOV, NOT, AND, XOR and rotations.
- The size is small and fits in the registers of most processors.
- There is no need to implement the inverse of the permutation.
- It needs only one pass for encryption and authentication.

### 2.3.2 Limitations

- It processes its input sequentially, that is it cannot be parallelized like, for instance, OCB.
- It loses its security under nonce reuse.
- Recovery of the inner state implies recovery of the key.
- Its S-box is more complex than those of other lightweight ciphers that have less algebraic degree.

# 3 The hash function Coral

## 3.1 Mode of operation

First the 256-bit state is initialized with all zeros. Then, as in classical sponges, there is an absorbing phase and a squeezing phase.

In the first phase the message is padded and absorbed in $r$-bit chunks. After each absorption the state is transformed via the permutation $\pi_I$.

The padding consists in a single bit with value 1 and just enough zeros to make their lengths a multiple of $r$ bits. In practice we assume all inputs are byte strings, so the padding consists of the byte valued 0x80 and enough zero bytes to make the length multiple of $r/8$.

The squeezing phase begins after the last block of input is processed. In each step of the squeezing $r$ bits of output are produced, with an application of the permutation between succesive outputs. As the rate is 32 bits, and the hash length is 256, we need a total of 8 applications of $\pi_I$.

The procedure is illustrated in figure 6 and specified in figure 7.

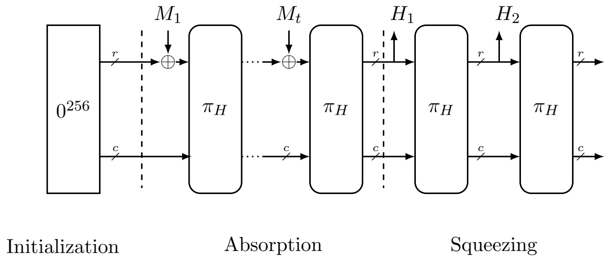

Figure 6: Hash function

```
HASH($M$)
// Initialization
$S = 0^{256}$
// Message processing
$t = \lfloor |M|/r \rfloor + 1$
$M_1 \ldots M_t = \mathrm{PAD}(M)$
for $i = 1$ to $t$
    $S_r = S_r \oplus M_i$
    $S = \pi_I(S)$
// Finalization
$n = |H|/r$ // $|H|$ is a multiple of $r$
for $i = 1$ to $n - 1$
    $H_i = S_r$
    $S = \pi_I(S)$
$H_n = S_r$
return $H$
```

Figure 7: Hash procedure

## 3.2 Security Claims

We claim 112 bits of security as per the $c/2$ security bound of the sponge construction. In accordance to the NIST call, we do not distinguish between messages one of which is a truncation of the other by a number of bits less than eight.

## 3.3 Advantages and limitations

### 3.3.1 Advantages

- It uses well known and tested primitives.

- It allows efficient bit-sliced implementations in software. All operations are defined in term of 64-bit word operations such as MOV, NOT, AND, XOR and rotations.
- The size is small and fits in the registers of most processors.

### 3.3.2 Limitations

- Its S-box is more complex than those of other lightweight ciphers that have less algebraic degree.

# 4 The permutations

We define four permutations, $\pi_I$, $\pi_{AD}$, $\pi_{AE}$ and $\pi_F$ which differ in the round constants and in the number of rounds (shown in table 1).

| Permutation | Rounds |
|---|---|
| $\pi_I$ | 10 |
| $\pi_{AD}$ | 6 |
| $\pi_{AE}$ | 6 |
| $\pi_F$ | 6 |

Table 1: Number of rounds

The permutations consist in a sequence of rounds. Each round is the composition of three transformations:

$$LD \circ \mathcal{S} \circ C$$

where $C$ is the addition of a round constant, $\mathcal{S}$ is a substitution layer, and $LD$ is a linear diffusion layer.

## 4.1 Addition of round constants

The first step of the permutation is the addition of a round constant to the second word of the state.

$$x_1 = x_1 \oplus C_i^\pi$$

The value of the round constants can be calculated as the exclusive-or of a permutation constant and the round number times 15.

$$C_i^\pi = C^\pi \oplus 15 \times i \quad \text{for } i \in \{0 \ldots 15\}$$

where the first round is round zero, $i$ is the round number and $C^\pi$ is the permutation constant (shown in table 2).

Table 3 shows the eight least significant bits of the round constants expressed as hexadecimal numbers (the 56 most significant bits are all zero). Note that not all the permutations require the same number of rounds, and no permutation in this specification requires 16 rounds.

| **Permutation** | **Constant** |
|---|---|
| $C^{\pi_I}$ | 0xBE |
| $C^{\pi_{AD}}$ | 0xAD |
| $C^{\pi_{AE}}$ | 0xAE |
| $C^{\pi_F}$ | 0xEB |

Table 2: Permutation constants

| **Round** | $\pi_I$ | $\pi_{AD}$ | $\pi_{AE}$ | $\pi_F$ |
|---|---|---|---|---|
| 0 | be | ad | ae | eb |
| 1 | b1 | a2 | a1 | e4 |
| 2 | a0 | b3 | b0 | f5 |
| 3 | 93 | 80 | 83 | c6 |
| 4 | 82 | 91 | 92 | d7 |
| 5 | f5 | e6 | e5 | a0 |
| 6 | e4 | f7 | f4 | b1 |
| 7 | d7 | c4 | c7 | 82 |
| 8 | c6 | d5 | d6 | 93 |
| 9 | 39 | 2a | 29 | 6c |
| 10 | 28 | 3b | 38 | 7d |
| 11 | 1b | 08 | 0b | 4e |
| 12 | 0a | 19 | 1a | 5f |
| 13 | 7d | 6e | 6d | 28 |
| 14 | 6c | 7f | 7c | 39 |
| 15 | 5f | 4c | 4f | 0a |

Table 3: Round constants

## 4.2 The substitution layer

The substitution layer consists in the parallel application of 64 identical 4-bit S-boxes. For this application the state is viewed as 64 bit-slices, with the word $x_0$ providing the most significant bit and $x_3$ providing the least significant one.

The S-Box is shown in table 4

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{S}(x)$ | 4 | 7 | 1 | 12 | 2 | 8 | 15 | 3 | 13 | 10 | 14 | 9 | 11 | 6 | 5 | 0 |

Table 4: The Yarará and Coral S-box

The algebraic normal form of the corresponding functions for the bit sliced application is as follows:

$$\begin{aligned}
x_0' &= x_0 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_0x_2x_3 \oplus x_1x_2x_3 \\
x_1' &= 1 \oplus x_1 \oplus x_2 \oplus x_0x_2 \oplus x_0x_3 \oplus x_2x_3 \oplus x_0x_1x_2 \oplus x_0x_2x_3 \\
x_2' &= x_1 \oplus x_3 \oplus x_0x_2 \oplus x_2x_3 \oplus x_0x_1x_3 \oplus x_0x_2x_3 \\
x_3' &= x_0 \oplus x_2 \oplus x_3 \oplus x_1x_3 \oplus x_0x_1x_2 \oplus x_0x_1x_3
\end{aligned}$$

### 4.3 The linear diffusion layer

The linear diffusion layer has three steps. The first and last step provide diffusion within each 64-bit word of the state, while the step in the middle mixes the four words.

The first and third steps, MixRows, consist in the application of four functions $\Sigma_0, \ldots \Sigma_3$, with the same constants used in the first four equivalent functions used in Ascon [5].

$$\begin{aligned}
\Sigma_0(x_0) &= x_0 \oplus (x_0 >>> 19) \oplus (x_0 >>> 28) \\
\Sigma_1(x_1) &= x_1 \oplus (x_1 >>> 61) \oplus (x_1 >>> 39) \\
\Sigma_2(x_2) &= x_2 \oplus (x_2 >>> 1) \oplus (x_2 >>> 6) \\
\Sigma_3(x_3) &= x_3 \oplus (x_3 >>> 10) \oplus (x_3 >>> 17)
\end{aligned}$$

The second step, MixColumns, is a left-multiplication by an almost MDS matrix as used in the MixColumns step of FIDES [3]. Its coefficients are restricted to 0 and 1, so it can be efficiently implemented with only XOR operations. The matrix is

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

The diffusion can be expressed, then, as

$$\textsc{MixRows} \circ \textsc{MixColumns} \circ \textsc{MixRows}$$

# 5 Security Analysis

$4 \times 4$ S-boxes have been extensively analyzed [7, 8, 9], so we selected one with strong cryptographic properties.

The S-box has the following properties:

- It's optimal against differential cryptanalysis, with a maximum differential probability $p = 1/4$.
- It's optimal against linear cryptanalysis, with bias $\epsilon = 1/4$.
- No single-bit difference in input results in a single-bit output difference.
- Its branch number is 3.
- All of its boolean functions have algebraic degree 3.
- It is PE equivalent, and hence affine equivalent, with the S-box S3 of Serpent.

The strong diffusion increases significantly the number of active S-boxes. For all inputs of up to 5 active S-boxes produces at least 20 active S-boxes in two rounds.

## 5.1 Differential and Linear Attacks

The only nonlinear component of Yarará and Coral is the S-box.

The S-box has been chosen to have the best possible differential an linear probabilities for its size, that is $2^{-2}$. This implies that we need 64 active S-boxes to ensure immunity against both differential and linear attacks.

The diffusion produces an avalanche effect that makes the number of active S-boxes exceed this number in very few rounds.

The following table shows the number of output S-boxes for inputs with 1 to 5 S-Boxes, of all possible weights.

| **Input** | **Output** |
|---|---|
| 1 | 19 |
| 2 | 19 |
| 3 | 19 |
| 4 | 18 |
| 5 | 15 |

For the inverse of the diffusion, no input of 5 or less S-boxes produces less than 40 active S-boxes in its output.

That means the lower bound for three rounds is 18 active S-boxes, assuming there is a 6 active S-boxes input that produces a 6 active S-boxes output. We consider that unlikely. Our heuristic searches have not found anything near that bound.

Using the heuristic search tool proposed by Dobraunig et al [4] the best linear trail we have been able to found for 3 rounds has 62 active S-boxes with a bias of $2^{-77}$.

So, given the optimal differential and linear characteristics of the S-box, together with the strong diffusion, added to the fact that an attacker has only access to 64 bit of the state in each step, we are confident that no differential or linear attack is possible.

## 5.2 Algebraic attacks

Algebraic attacks like zero-sum and cube attacks depend on the degree of the boolean functions of the S-box and the number of rounds. In our case, the S-box has the maximum possible degree in all four functions, and the number of rounds has been selected so the algebraic degree of the permutation is high enough ($3^6 = 729 > 256$).

We expect these kind of attacks to be difficult to mount, and in the light of the results obtained for Ascon and other ciphers we expect that even if such attack is mounted against the indifferentiability of the permutation, it will not escalate against the security of the cipher itself.

## 5.3 Related keys and related nonces

Some attacks use keys or nonces that are related to improve the differential probabilities of an attack. To prevent this, in Yarará the initial permutation is stronger and both the key and the nonce are passed through 10 rounds before being allowed to interact with any data.

# 6 Design Rationale

The requirements specified by NIST target a 112-bit security level for both algorithms.

In the case of the AEAD algorithm, the NIST call states a minimum key length of 128 bits and a minimum nonce length of 96 bits (at least for the family primary member).

In the case of a sponge based hash function, working under the $c/2$ security bound, the 112-bit security level requires a capacity of at least 224 bits.

These restrictions led us to choose a 256-bit state. This choice leads to the possibility of using bit-sliced $4 \times 4$ S-boxes in a permutation based cipher.

For Yarará the size of the state permits 128 bits of security under the $c - a$ bound proved in [1] when the data complexity is limited to $2^a$ $r$-bit blocks. As NIST demands the ability to process up to $2^{50} - 1$ bytes under one key, that is $2^{47}$ 64-bit blocks, 192 bits of capacity are enough to reach the desired security level.

For Coral, the 256 bit state allows 112 bits of security with a rate of 32 bits and a capacity of 224 bits.

We decided to use well known primitives. The S-box is PE equivalent to one of the Serpent S-boxes (S3). Not only is optimal from the point of view of differential and linear cryptanalysis, but it also has algebraic degree 3, which makes it resistant to cube attacks and interpolation attacks.

The diffusion has mixed layers, implying a loss in speed, but we decided to err on the side of security.

The sponge construction has been extensively analyzed, and in fact has been chosen for the NIST Standard SHA3.

Different constants have been selected for each permutation to ensure domain separation. Additionally, there is a bit added between AD and plaintext processing to further harden this domain separation.

The constants selected are considered adequate to thwart slide attacks, and they have a simple structure based on obvious initial values to prove there are not used in hidden backdoors.

# 7 Bibliography

## References

[1] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. On the security of the keyed sponge construction.

[2] Bertoni, G., Daemen, J., Peeters, M., and Assche, G. V. Permutation based encryption, authentication and authenticated encryption, 2012.

[3] Bilgin, B., Bogdanov, A., Knežević, M., Mendel, F., and Wang, Q. Fides: Lightweight authenticated cipher with side-channel resistance for constrained hardware. In *Cryptographic Hardware and Embedded Systems - CHES 2013* (Berlin, Heidelberg, 2013), G. Bertoni and J.-S. Coron, Eds., Springer Berlin Heidelberg, pp. 142–158.

[4] Dobraunig, C., Eichlseder, M., and Mendel, F. Heuristic tool for linear cryptanalysis with applications to caesar candidates. In *Advances in Cryptology – ASIACRYPT 2015* (Berlin, Heidelberg, 2015), T. Iwata and J. H. Cheon, Eds., Springer Berlin Heidelberg, pp. 490–509.

[5] Dobraunig, C., Eichlseder, M., Mendel, F., and Schläffer, M. Ascon v1.2. Submission to the CAESAR competition: `http://competitions.cr.yp.to/round3/asconv12.pdf`, 2016.

[6] G. Bertoni, J. D. M. P., and Assche, G. V. Sponge Functions. In *Ecrypt Hash Workshop 2007* (2007).

[7] Leander, G., and Poschmann, A. On the classification of 4 bit s-boxes. In *Arithmetic of Finite Fields* (Berlin, Heidelberg, 2007), C. Carlet and B. Sunar, Eds., Springer Berlin Heidelberg, pp. 159–176.

[8] Saarinen, M.-J. O. Cryptographic Analysis of All 4 x 4-Bit S-Boxes. In *Selected Areas in Cryptography* (Berlin, Heidelberg, 2012), A. Miri and S. Vaudenay, Eds., Springer Berlin Heidelberg, pp. 118–133.

[9] Zhang, W., Bao, Z., Rijmen, V., and Liu, M. A new classification of 4-bit optimal s-boxes and its application to present, rectangle and spongent. In *IACR Cryptology ePrint Archive* (2015).