RAMus - a New Lightweight Block Cipher for RAM Encryption

Raluca Posteuca[1] and Vincent Rijmen[1,2]

[1] imec-COSIC, Department of Electrical Engineering (ESAT), KU Leuven, Belgium
{raluca.posteuca, vincent.rijmen}@esat.kuleuven.be
[2] Department of Informatics, University of Bergen, Bergen 5020, Norway

Keywords: RAMus· RAM encryption · Branch number · 2S-strategy

Abstract. Over the past decades, there has been a dramatic increase of the attacks recovering the data from the RAM memory. These have heightened the need for new solutions and primitives suitable for the encryption of this information. In this paper we introduce RAMus, a new tweakable lightweight block cipher whose properties support its usage for securing the RAM memory. In this sense, RAMus attains all the requirements provided by the (German) Federal Office of Information Security (BSI) in the domain of encryption algorithms suitable for RAM and memory encryption. The design strategy of RAMus is inspired from the LS-approach. Compared to the literature, in our proposal the linear layer is replaced by a second Sbox layer. In RAMus, the diffusion is ensured by the Sbox layers, which use Sboxes with a non-trivial branch number.

1 Introduction

The security of a personal device, such as a mobile phone or a computer, is one of the most analyzed topics in the domain of cryptography, security, and privacy. Even though most of the vulnerabilities arise from the online behavior of the device's user, in the last decade special attention was given to the security of the data stored in the memory of the device. The attacks aiming at recovering the data from the RAM, such as the cold boot attack (26) or the direct memory access attack (40), proved the increasing importance of protecting this information. In order to increase the security of the data stored in the RAM memory, both the academia and the industry invested their resources in designing a series of solutions. While the resulted proposals employ different techniques to ensure the security of the cryptographic secrets, most of them are based on one of the following block ciphers: AES (23), Prince (19), Qarma (10) or ASCON (24).

In order to support the development of RAM encryption solutions, The (German) Federal Office of Information Security (BSI) published, in 2013, a methodology for cryptographic rating of memory encryption schemes used in smartcards and similar devices (1). According to this methodology, an algorithm

suitable for memory encryption schemes should exhibit small area for its implementation, while having high speed, which is translated in our work by both low latency and high throughput. Furthermore, the methodology recommends the use of a tweakable block cipher, where the tweak is parameterised with the memory address of the plaintext. Last but not least, the methodology discusses the necessary security of a suitable block cipher with respect to the known attacks. While linear and differential cryptanalysis are considered critical, the methodology considers less relevant the security against related-key attacks. The methodology also discusses the impact of side-channel attacks, noting that this type of attacks could lead to critical vulnerabilities in some particular scenarios.

Our contribution In order to contribute to the efforts of the cryptographic community in the area of secure memory encryption, we propose RAMus, a new lightweight tweakable block cipher. The design of RAMus follows the *2S-strategy*, a new design framework introduced in this paper. The cipher satisfies the constraints imposed by the BSI methodology, while ensuring that its *threshold implementation* against side-channel attacks does not have any overheads. RAMus is a tweakable block cipher, with a tweak space of 2^{64}. Therefore, for the 32- and 64-bit systems, it allows the usage of the same symmetric key for the entire RAM memory, without the vulnerabilities induced by the ECB mode of operation and the overhead of other modes of operation, such as XEX or XTS.

In order to ensure low area of the implementation of RAMus, the round function, tweak update function and key schedule use only two basic operations: one 8-bit Sbox and the XOR addition. To ensure good diffusion through the cipher, the Sbox was designed such that it has good cryptographic properties, while having non-trivial linear and differential branch numbers.

The latency of the cipher is closely related to the number of non-linear operations used for the round function. We note that the chosen Sbox can be implemented with only 8 non-linear gates, the minimum number of non-linear gates of an 8-bit Sbox with good cryptographic properties. Moreover, the strategy used for the design of the Sbox ensures resistance against side-channel attacks according to the literature (20; 32).

Related work The related work covers two areas. The first one regards the block ciphers that are used for RAM encryption solutions, while the second one refers to the design and usage of Sboxes with non-trivial linear and differential branch number.

Block ciphers suitable for RAM encryption. The most common block ciphers used to design RAM encryption solutions include AES, Prince, Qarma and ASCON. Although the security of all these ciphers was subjected to extended analysis (especially in the case of AES, which was selected as a NIST standard, and ASCON, which is part of the final portofolio of the CAESAR competition and a finalist of the NIST Lightweight competition), we remark that none of these ciphers fulfill all the requirements presented in the BSI methodology. Note

that AES, Prince and ASCON are not tweakable, while the Sbox of Qarma does not lend itself easily to a lightweight side-channel resistant implementation.

Sboxes with non-trivial branch numbers. Through the last decades, the problem of linear and differential branch numbers of an Sbox caught the attention of the cryptographic community. The first step in this direction regards the design and analysis of 4-bit Sboxes with non-trivial differential branch number, such as the Sboxes of Serpent (15) or PRESENT (18). The natural extension of this field was proposed in (36), which presents the classification of all 4-bit Sbox equivalence classes with respect to the differential branch number. The next step of this research was the design and analysis of 5-bit Sboxes with non-trivial branch numbers, such as the Sbox of ASCON which has both linear and differential branch number 3. This work was continued in (37), which proposes new design strategies for 5-bit and 6-bit Sboxes with non-trivial branch numbers.

Further, (28) introduces the unbalanced-bridge approach, a technique suitable for the design of 8-bit Sboxes with linear and differential branch number 3. Such Sboxes have good cryptographic properties, while allowing for bit-slice implementations which use at least 11 non-linear gates. Using this type of Sboxes, the authors propose the block cipher PIPO, which follows the LS-design framework. We note that the authors also present a series of 8-bit Sboxes with differential branch number 4, but with non-linearity 0, concluding that this type of Sboxes would induce vulnerabilities with respect to linear cryptanalysis.

Structure of the paper The rest of this paper is organized as follows: in Section 2 we present some terminology regarding linear and differential cryptanalysis, the branch number of an Sbox and the LS-design framework. In Section 3 we introduce the 2S-strategy, a new design technique inspired by the LS-design framework. Section 4 introduces the RAMus block cipher, as a parameterization of the 2S-strategy, and Section 5 discusses the design rationale of RAMus. Section 6 presents the security analysis of RAMus with respect to the most important cryptanalytic techniques, while in Section 7 we discuss the performance of RAMus in hardware implementations.

2 Preliminaries

Linear cryptanalysis Linear cryptanalysis (31) was introduced in 1993 by Matsui as an attack against DES (6). This approach aims at finding linear approximations between the bits of the plaintext and the ciphertext which hold with a probability different from . In order to exploit such approximations, it is necessary that the associated probability is different from 0.5, i.e.

$$Pr = \#\{p \in \mathbb{F}_2^n | \bigoplus_i p_i \oplus \bigoplus_j c_j = 0\}/2^n \neq 0.5,$$

where p_i and c_j represent the i^{th} bit of the plaintext and the j^{th} bit of the ciphertext, respectively. The quality of a linear approximation defines the success

rate of the future attack, and in this paper it is measured by the correlation of the linear approximation, which is defined as $corr = 2 \cdot Pr - 1$.

The most common approach to finding such approximations is by using a divide-and-conquer approach. Matsui's strategy was to identify linear approximations between the input and the output of each operation and further connect them, resulting in the *linear trail*. The analysis of every operation can be performed by following the rules of propagation of linear trails introduced in (14; 21; 22). If the case of the propagation through linear layers can be considered straightforward, the case of non-linear layers is more involved. In particular, for the analysis of the Sbox with respect to linear cryptanalysis the standard approach is to compute the corresponding Linear Approximation Table (LAT).

Definition 1. *Let S be an Sbox of size n and $"\cdot"$ be the standard inner product. Then, the LAT of an Sbox S represents the matrix defined as follows:*

$$LAT_S[\alpha, \beta] = \#\{x | \alpha \cdot x \oplus \beta \cdot S(x) = 0\} - 2^{n-1}.$$

The maximum absolute value of LAT_S is called the *liniar uniformity* of the Sbox S and it defines the quality of the Sbox with respect to linear cryptanalysis.

Differential cryptanalysis Differential cryptanalysis (16) was introduced in 1990 by Biham and Shamir, also as an attack on DES. This attack aims at analysing the propagation of differences from a plaintext pair to the corresponding ciphertext pair. The approach is similar to the one presented above, involving the analysis of the propagation through the particular operations of a cipher and further connect them, resulting in the *differential characteristic*. Usually, the difference is considered with respect to the XOR operation and in this paper we conform to this. While the difference propagation through the linear layers are straightforward, the analysis of the differences' propagation through a non-linear layer involves the computation of its corresponding Differential Distribution Table (DDT).

Definition 2. *The DDT of an Sbox S represents the matrix of integers defined as follows:*

$$DDT_S[\delta, \Delta] = \#\{x | \Delta = S(x) \oplus S(x \oplus \delta)\}.$$

The maximum value of DDT_S is called the *differential uniformity* of the Sbox S and it defines the quality of the Sbox with respect to differential cryptanalysis.

Branch number The main criteria in the design of a block cipher is represented by the properties of confusion and diffusion. In most of the ciphers from literature, the confusion is ensured by the choice of the non-linear (Sbox) layer, while the diffusion is often ensured by the linear layer(s) of the cipher. The most common technique to measure the diffusion of a cipher is given through the means of the branch number of the underlying operations. For the purposes of this work, we only discuss the concept of the branch number associated to an Sbox. Let us denote the Hamming weight of a byte x by $wt(x)$.

Table 1. The properties of the DDT and the LAT of an Sbox

Any Sbox	Invertible Sbox
If $DDT(0, \Delta) \neq 0$, then $\Delta = 0$.	If $DDT(\delta, 0) \neq 0$, then $\delta = 0$.
If $LAT(\alpha, 0) \neq 0$, then $\alpha = 0$.	If $LAT(0, \beta) \neq 0$, then $\beta = 0$.

Definition 3. *The differential and linear branch numbers of an Sbox S, denoted by BN_d and BN_l respectively, are computed as:*

$$BN_d(S) = \min\{\mathrm{wt}(\delta) + \mathrm{wt}(\Delta) | DDT_S(\delta, \Delta) \neq 0\}$$
$$BN_l(S) = \min\{\mathrm{wt}(\alpha) + \mathrm{wt}(\beta) | LAT_S(\alpha, \beta) \neq 0\}$$

Note that the LAT and the DDT of an Sbox have the properties described in Table 1. A consequence of these properties is that the minimum value of the linear and differential branch number of an invertible Sbox is 2, therefore we consider this to be the trivial linear and differential branch number. Frequently, the design of the non-linear layer consists of independent, parallel applications of one or more Sboxes on partitions of the internal state. In this case, the branch number of the entire non-linear layer is given by the lowest branch number of the underlying Sboxes.

Bounds on linear and differential branch number. In (38) the authors present the bound on linear and differential branch number of permutations. We summarize their results in Lemma 1.

Lemma 1. *Let $S : \mathbb{F}_2^n \to \mathbb{F}_2^n$ be a non-linear permutation. Then,*

- $BN_l(S) \leq n - 1$
- *if* $n = 4$, $BN_d(S) \leq 3$
- *if* $n \geq 5$, $BN_d(S) \leq \lceil 2\frac{n}{3} \rceil$

In particular, the maximum differential branch number for an 8-bit Sbox is 6, while the maximum linear branch number is 7. (30) presents a technique of designing non-linear layers with maximum differential branch number. The goal of this paper was to introduce new non-linear diffusion layers, therefore the resulting permutations have trivial linear and differential uniformity.

The LS-design framework Nowadays, one of the goals of cryptographers is to analyse and propose different design strategies meant to ensure the security of the future symmetric primitives against the most important attacks. In order to ensure the resistance of a cipher against the two most significant mathematical attacks, namely linear and differential cryptanalysis, Daemen and Rijmen proposed the wide-trail strategy (23).

In the last decades, the scientific community focused on analysing the security of a block cipher against side-channel attacks (27). One of the approaches to ensure the security of a cipher against such attacks is to implement the so called

masking techniques (34). While these techniques can ensure the needed level of security, they highly influence the costs of implementing a block cipher. In order to address this issue, Grosso et al. introduced the LS-design framework (25).

The internal state of a cipher based on the LS-design framework is viewed as a $l \times s$ matrix. The round function, apart from the key and constant addition, consists of two operations: a non-linear layer defined by the parallel application of an Sbox on each row of the matrix, and a linear layer in which a linear function is applied independently on each column. For more details regarding the properties of these two operations and possible parameterizations we refer the reader to the original paper.

3 The 2S-strategy

In this paper we introduce the *2S-strategy*. The aim of this strategy is to lead to the design of a tweakable block cipher which is designed only by using non-linear operations. This design strategy is inspired by the LS-strategy described in Section 2. In order to ensure the diffusion through the cipher, we design non-linear layers with non-trivial linear and differential branch numbers.

3.1 Notations

The internal state of a cipher based on the 2S-strategy is viewed as an $r \times c$ matrix of bits. While the values of r and c can be chosen by the designer, in this paper, we consider $r = 8$ and $c = 8$, therefore the internal state contains 64 bits, indexed as described in Figure 1. In this paper we consider the rows indexed top-to-bottom, while the columns are indexed left-to-right, i.e. the first row and the first column are the ones containing the bit indexed 1. For the bytes composition, we use the big-endian order. More specifically, for the first row the bit in position 8 represents the least significant bit, while in the first column the bit in position 1 represents the most significant one.

Throughout this paper we would refer to two manners in which a byte array could be extracted from the 8×8 matrix of bits s. We denote by $v_R(s)$ the array containing the bytes composed by the rows of the matrix, while $v_C(s)$ denotes the bytes read at a column level. More precisely, the first value of $v_R(s)$ is the byte composed by the bits indexed from 1 to 8, while the last component of the $v_C(s)$ is represented by the bits indexed by the values multiple of 8. A description of how the arrays $v_R(s)$ and $v_C(s)$ are obtained from the internal state s is described in Figure 1.

We also introduce the inverse operations of v_R and v_C, which take a byte array as input and return an 8×8 matrix, as follows:

$$s^R(v_R(s)) = s, \qquad s^C(v_C(s)) = s$$

3.2 The round function

The round function of the cipher is described by two Sbox layers, one tweak and one key addition. More precisely, the round function assumes the following

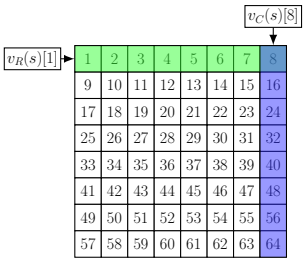

Fig. 1. The indexes of the internal state and the $v_R(s)$ and $v_C(s)$ functions' application.

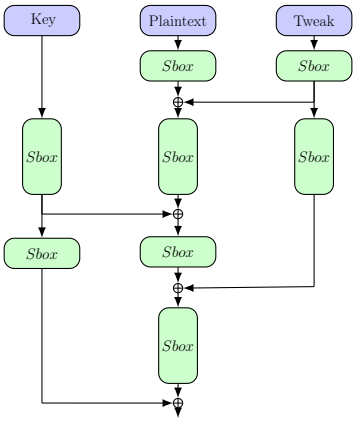

Fig. 2. Two consecutive rounds of the cipher. The lying rectangles represent the first non-linear layer, while the standing rectangles represent the second one.

operations: first, an Sbox S is applied on every row of the internal state; secondly, the round tweak is added to the internal state by the use of bitwise XOR; thirdly, the same Sbox S is applied on every column of the internal state; finally, the round key is added to the internal state. The description of the first two rounds of such a cipher is depicted in Figure 2. We underline that, depending on the use case, more than one Sbox could be used in order to design a block cipher based on the 2S-strategy. However, the use of multiple Sboxes could have several drawbacks. Firstly, the chosen Sboxes must be designed such that they exhibit non-trivial linear and differential branch numbers, while having non-trivial uniformities. Secondly, the use of different Sboxes leads to an increase in the area needed for the implementation of the cipher.

The Sbox layers The diffusion of most ciphers in the literature is ensured by using linear layers with non-trivial branch numbers. In the case of the 2S-strategy, both the confusion and the diffusion of the cipher are ensured by using two non-linear layers in each round of the cipher. Both layers assume the parallel application of an Sbox on partitions of the internal state. For the first layer, denoted SB_R, the Sbox is applied on each row independently, while for the second layer, denoted SB_C, the Sbox is applied on each column. The index denotes the manner in which the inputs were chosen, where the indexes "R" and "C" marks the appliance of the Sbox on rows and columns, respectively. Figure 3 and Figure 4 describe the manner in which the non-linear layers SB_R and SB_C are applied. We note that the two layers are affine equivalent functions, defined by the relation $SB_R(s) = SB_C(s^T)^T$, where s^T denotes the transposition of the matrix s.

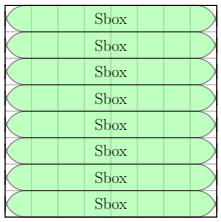

Fig. 3. The layer SB_R.

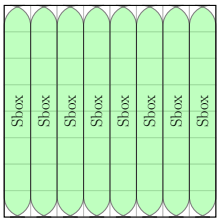

Fig. 4. The layer SB_C.

In order to ensure the security of a cipher based on the 2S-strategy, the chosen Sboxes need to have good cryptographic properties, such as good linear and differential uniformity. Moreover, in order to ensure the diffusion through the non-linear layer, the Sboxes must also have non-trivial linear and differential branch numbers. In Section 5 we propose a parameterization for the Sbox S, we discuss the properties of an Sbox suitable for the 2S-strategy and we describe a design strategy that leads to the design of an Sbox with the suitable properties.

Key schedule and tweak update function In order to ensure a small area of the hardware implementation of a cipher based on the 2S-strategy, we designed the key schedule and the tweak update function by using the same non-linear layers as the round function. The choices of these two functions determine the efficiency of the cipher in practice, in the use scenario. Usually, the encryption of the RAM data is performed using the same symmetric key, therefore the key derivation function can be performed only once, in the initialization phase of the system. In this phase, all the round keys can be computed and stored in a secure register or device.

In order to ensure a higher resistance of a cipher based on the 2S-strategy against linear, differential and related-key attacks, we designed the key schedule and the tweak update function by using one non-linear layer in each round. Moreover, we designed these functions such that in two consecutive rounds different non-linear layers are applied.

The tweak update function is designed as follows: in the odd indexed rounds, the Sbox is applied on each row of the current tweak, while in the even numbered rounds, the Sbox is applied on every column. Note that the first round is indexed by 1. A pseudocode of this function is given in Algorithm 1.

Algorithm 1: The tweak update function for round r

Result: The round tweak
if *((r % 2) == 1)* **then**
  $tweak_r = SB_R(tweak_{r-1})$;
else
  $tweak_r = SB_C(tweak_{r-1})$;
end

Algorithm 2: The key schedule for the round r

Result: The round key
if *((r % 2) == 0)* **then**
  $key_r = SB_R(key_{r-1})$;
else
  $key_r = SB_C(key_{r-1})$;
end

A similar approach is used for designing the key schedule, where the operations on the even and odd rows are performed in reverse. More precisely, in the odd indexed rounds, the Sbox is applied on the columns of the current key, while in the even indexed rounds the Sbox is applied on the rows.

A pseudocode of the key schedule algorithm is described in Algorithm 2.

Round constants The round constants could be added either on the round function, the tweak update function or the key schedule, depending on the goals, in terms of performance, of the new designed block cipher. While in Section 4 we propose an algorithm for the generation of the round constants, we entrust the future designers to create personalized algorithms which accomplish this purpose.

4 The description of RAMus

The RAMus cipher represents a practical parameterization of the 2S-strategy. RAMus is a tweakable lightweight block cipher with 64 bit block and 128 bit key and 17 rounds. The cipher attains all the requirements provided by the (German) Federal Office of Information Security (BSI) in the domain of encryption algorithms suitable for RAM and memory encryption.

The Sbox layers As mentioned in Section 3.2, an Sbox suitable for the 2S-strategy needs to have good cryptographic properties, together with non-trivial linear and differential branch numbers. In Section 5 we propose a design strategy which could be used to generate Sboxes that fulfill all the necessary properties. By following this strategy, we designed an Sbox which has differential branch number 4 and linear branch number 3, while both the linear and differential uniformities are 64. To the best of our knowledge, this is the first published Sbox with differential branch number 4 and non-trivial linear and differential uniformities.

Moreover, given the use case of RAMus, in the design of the Sbox S we used a supplementary constraint regarding the optimization of Sbox's implementation with respect to the number of non-linear gates. We discuss the design rationale of the Sbox in Section 5, while the full description of the Sbox, given through a look-up table, is given in the Appendix, in Table 4.

Key schedule, tweak and round constants addition The 2S-strategy defines a key schedule in which the key has the same length as the block - in the case of RAMus 64 bits. In order to design a block cipher with a master key of length 128, we use the same approach introduced by the authors of the Prince cipher (19). More precisely, we split the 128-bit master key k into $k = (k_0, k_1)$, where k_0 and k_1 represents the first and the last 64 bits of the key, respectively. The key is then extended from 128 bits to 192 bits as follows:

$$(k_0, k_1) \rightarrow (k_0, (k_0 \gg 1) \oplus (k_0 \gg 63), k_1),$$

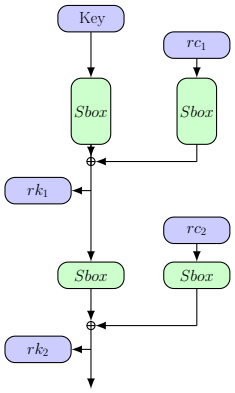

Fig. 5. The first two rounds of the key schedule of RAMus.

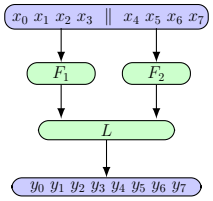

Fig. 6. The SPN design strategy for Sboxes

where $x \gg y$ defines the circular shift of the 64-bit word x with y positions. The first two keys are used for the initial and final whitening, while the key k_1 represents the input of the key schedule of RAMus.

In order to minimize the storing space and the latency of the cipher, we chose to add the round constants to the key schedule, instead of adding them to the round function. The constants are generated depending to the current round index, and they are obtained after applying one Sbox layer. A pseudocode of the key schedule is described in Algorithm 3. We note that, for the first round, the key rk_{r-1} represents the master key of the cipher.

In each round i, the constants rc_i are added to 4 different and consecutive rows or columns, depending on the round index. More precisely, in the odd indexed round $2 \cdot r+1$, the constants are added to the columns $r, r+1, r+2, r+3$, while in the rounds $2 \cdot r$, the constants are added to the rows $r, r+1, r+2, r+3$. Therefore, in every round, four constant bytes are added to the internal state of the key schedule. These bytes are computed by applying the Sbox on the values $4 \cdot r$, $4 \cdot r+1, 4 \cdot r+2$ and $4 \cdot r+3$. Figure 5 describes the first two rounds of the key schedule of RAMus.

The tweak update function follows accurately the corresponding description of the 2S-strategy, which is described in Section 3.2 and in Algorithm 1.

5 Design rationale

Linear vs. non-linear layers The linear layer is considered a core component of the ciphers based on the SPN, LS or ARX strategies. Many papers aimed at introducing different strategies for the design of the linear layer, both from a security and efficiency perspective (e.g. (23), (7)). The general goal of the linear layer is to ensure the diffusion through the round function. Usually, the design of optimal linear layers assumes the search of linear functions that have a high branch number, while allowing for an efficient implementation. Recent works

Algorithm 3: The key schedule - computing the key for the r^{th} round

Input: The $(r-1)^{th}$ round key rk_{r-1}
Output: The r^{th} round key rk_r
$x_r = r/2$;
// Defining the round constants' associated initial array $RC[8]$
for $i = 0$ **to** 3 **do**
  $RC[(x_r + i)\%8] = 4 \cdot x_r + i$;
end
// Computing the round constant rc_r and the round key rk_r
if *((r % 2) == 1)* **then**
  $rc_r = SB_R(s^R(RC))$;
  $rk_r = SB_R(rk_{r-1}) \oplus rc_r$;
else
  $rc_r = SB_C(s^C(RC))$;
  $rk_r = SB_C(rk_{r-1}) \oplus rc_r$;
end

such as (28; 35) introduced methods to design Sboxes with non-trivial linear and differential branch number. The non-linear layer of RAMus represents a trade-off between the constraints of a good linear and a good non-linear layer, allowing for a design following the 2S-strategy (with no linear layers).

Sboxes with non-trivial branch number The design of the Sbox S followed three main goals. The first goal of our design strategy was to optimize the linear or the differential branch number of an Sbox, while ensuring the fact that the linear and differential uniformities are non-trivial. The second goal was to ensure, by design, the resistance of RAMus against power analysis, i.e. to ensure that RAMus has an efficient masked implementation. The third goal was to ensure the low latency of the cipher, in the masked implementation, therefore we aimed at optimising the number of non-linear gates of S. In order to ensure these three goals, our approach was to identify the design strategies for designing an Sbox with a known, masking-friendly, design strategy, as the ones presented in (20; 32). The Sbox S follows the SPN Sbox design strategy, which is depicted in Figure 6.

The main idea of this strategy is to divide the 8-bit input into two equal parts of 4 bit each, i.e. $x = x_1||x_2$. Then x_1 and x_2 are used as inputs for two 4x4 non-linear functions F_1 and F_2, respectively, resulting in $y = F_1(x_1)||F_2(x_2)$. The result y is then used as an input for an 8-bit linear layer L. Formally, the Sbox S can be described as $S(x_1||x_2) = L(F_1(x_1)||F_2(x_2))$.

While the SPN Sbox design strategy facilitates the design of Sboxes with resistance against power analysis, it has an important drawback: the linear and differential uniformities of the resulting Sbox are determined by the properties of F_1 and F_2. Let us denote by δ_f and l_f the differential and linear uniformity of the function f, respectively. Then: $\delta_S = 16 \cdot \max\{\delta_{F_1}, \delta_{F_2}\}, l_S = 16 \cdot \max\{l_{F_1}, l_{F_2}\}$.

Therefore, by using this approach the lowest linear and differential uniformity of S is 64.

Since one of the goals of our design strategy was to optimize the differential branch number, we analysed the properties of the non-linear functions F_1 and F_2, together with the properties of the linear layer L. Our design strategy is based on the following observation.

Observation 1 *Let L be the identity function. Then*

$$BN_d(S) = \min\{BN_d(F_1), BN_d(F2)\}$$

According to the literature (36), the highest differential branch number for a 4-bit Sbox is 3. Therefore, Observation 1 provides us a method to design 8-bit Sboxes with the non-trivial differential branch number 3.

In order to design S such that it is also optimised with respect to the number of necessary non-linear gates, we parameterize the functions F_1 and F_2 with the PRESENT Sbox, which can be implemented with only 4 non-linear gates, according to (33). Note that this is in fact the minimum number of non-linear gates that can be used in the implementation of any 4-bit Sbox with good cryptographic properties. Moreover, according to (17; 33), the Present Sbox also allows for a 3-share TI implementation.

For the design of the linear function L, we opted for a particular function with branch number 4 such that will ensure that the final differential branch number of S is 4. The linear function L was designed using three rotations, as follows $L(x) = \text{rot}(x, 1) \oplus \text{rot}(x, 2) \oplus \text{rot}(x, 5)$, where by $\text{rot}(x, i)$ we denote the circular left shift of the byte x by i positions. By using this design strategy, with different parameterizations for the non-linear function F_1 and F_2 and for the linear function L, different Sboxes with similar properties can be designed.

6 Security analysis of RAMus

Our analysis included a series of adversarial models, depending on the capabilities of the adversary. Depending on the attack scenario, the adversary can control the plaintext or the plaintext and the tweak (known- or chosen-plaintext attack), or he can even control the master key (related-key attack).

Given the use case of RAMus (RAM encryption solutions), the most suitable adversarial model is the one in which the adversary can fully control the plaintext. While the tweak value cannot be fixed, the adversary can choose the tweak difference, since the tweak represents the memory address associated to the plaintext. We consider less relevant the related-key scenario in this use case, since all the encryptions are performed using the same, fixed, master key.

6.1 Theoretical proven bound

In order to compute the theoretical upper bound of any differential characteristic or linear trail, we use the method introduced by the wide trail strategy. We

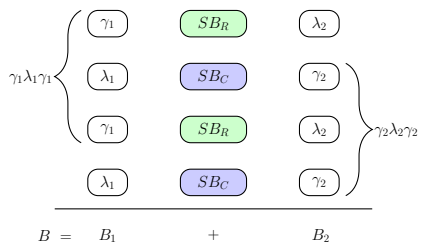

Fig. 7. The two associations between SB_R and SB_C to the γ and λ functions

bound the number of active Sboxes by using *The 2-round Propagation Theorem* provided by Daemen and Rijmen in (23).

Theorem 1 (The 2-Round Propagation Theorem). *For a key-alternating block cipher with a $\gamma\lambda$ structure, the number of active bytes of any two round trail is lower bounded by the (branch) number of λ.*

In (23), γ represents a local non-linear transformation, in which any output bit is influenced only by a set of input bits, while λ represents a linear mixing transformation with high diffusion. Classically, the γ function is represented by an Sbox layer, in which the Sbox is applied on partitions of the input bits, while the λ function is designed such that it has a high branch number. We underline that, since the number of active Sboxes is not influenced by the γ function, Theorem 1 in fact computes the number of active Sboxes of the $\gamma\lambda\gamma$ function. According to Section 5, the non-linear layer of RAMus satisfies both criteria: it represents a local non-linear transformation, while it has a non-trivial linear and differential branch number.

In order to compute the lower bound of the number of active Sboxes in 2 rounds of RAMus we apply Theorem 1 twice, with different correspondence between the γ and λ functions and the two non-linear layers SB_R and SB_C.

In order to compute the number of active Sboxes of the first non-linear layer, we identify γ to SB_R and λ with SB_C. According to the theorem, the number of active Sboxes of SB_R, in two rounds of RAMus is given by the branch number of SB_C. Accordingly, the number of active Sboxes of the second non-linear layer is bounded by the branch number of SB_R. Figure 7 describes these associations.

Therefore, the minimum number of active Sboxes in two rounds of RAMus can be computed as $B = B_1 + B_2$, where B_1 and B_2 represent the branch number of SB_R and SB_C respectively.

Since the branch number of both SB_R and SB_C are equal to the branch number of the Sbox S, the number of active Sboxes in two rounds of RAMus is equal to twice the branch number of S. Therefore, for 2 rounds of RAMus, the minimum number of active Sboxes is 8 for differential cryptanalysis and 6 for linear cryptanalysis. This analysis is performed in the fixed tweak scenario, in which the attacker can control both the plaintext and the tweak.

Table 2. The minimum number of active Sboxes, in different scenarios

Round nr.		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Fixed tweak	Diff sk	2	8	10	16	18	24	26	32	34	40	42	48	50	56	58	64	66
	Diff rk	1	6	9	14	17	22	25	30	33	38	41	46	49	54	57	62	65
	Lin	2	6	8	12	14	18	20	24	26	30	32	36	38	42	44	48	50
Not fixed tweak	Diff sk	2	6	10	15	18	22	26	31	34	38	42	47	50	54	58	63	66
	Diff rk	1	6	9	14	17	22	25	30	33	38	41	46	49	54	57	62	65
	Lin	3	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68

6.2 SAT-based analysis

The second step of our analysis was to use SAT-based methods to evaluate the security of RAMus against linear and differential cryptanalysis. In our analysis we used the ARXpy tool (11).

We performed our analysis in two main scenarios, depending on the capabilities of the adversary to control the tweak input. Therefore, in our first scenario the tweak is constant in all the encryptions, while for the second scenario the tweak is different, but the adversary can observe the values of the tweaks, and, therefore, their difference. For these scenarios, we analysed the propagation of differences in the single key (Diff sk) and related-key (Diff rk) scenarios, together with the propagation of the linear masks (Lin). The results are presented in Table 2. We note that the table presents the minimum number of possible active Sboxes after applying any number of rounds of RAMus between 1 and 17.

Moreover, we mention that all our experiments were performed by using a generic implementation table of the Sbox S, which does not represent a real Sbox, but imposes the constraints that its properties, such as the linear and differential branch numbers and uniformities, are the same as the Sbox S presented in Section 5. We stress that our experiments did not take into account neither the particular LAT, nor the DDT of the Sbox S. Therefore, the results obtained by using this approach represent only a lower bound of the number of active bits. In practice, the minimum number of active Sboxes could be higher.

The attentive reader will notice that, while the number of active Sboxes in the two scenarios are similar with respect to differential cryptanalysis, in the case of linear cryptanalysis the number of active Sboxes is quite different (favouring our use case). The reason for this difference can be explained by the different propagation of linear approximations through the XOR operation.

The number of active Sboxes, together with the particular cryptographic properties of the associated Sbox, leads to an estimation of the security level of the cipher against linear and differential cryptanalysis. In our analysis, we evaluate the security of RAMus by using the assumption that all the Sboxes are independent. From this point of view, our analysis follows the same approach used to assess the security of the CS-cipher (39; 42). Moreover, we mention that this assumption is also used to assess the security of a series of permutation-based algorithms, such as the NIST LWC submissions SpoC (8) or SPIX (9).

Resistance of RAMus against differential cryptanalysis. The most common approach in the security evaluation of a cipher against differential cryptanalysis

is to upper bound the probability p of any differential characteristic. If the bound is smaller than 2^{-k}, where k denotes the size of the key, then an attack based on differential cryptanalysis is not feasible. Under the independence assumption, the probability p is computed by $p = p_S^s$, where s represents the number of active Sboxes and p_S is the highest probability associated to an active Sbox. Note that p_S can be computed using the differential uniformity δ_S.

In the particular case of RAMus, $p_S = 2^{-2}$ and, according to Table 2, $s > 64$ in all the scenarios based on differential cryptanalysis. Therefore

$$p < (2^{-2})^{64} \Rightarrow p < 2^{-128}$$

Therefore, an attack based on differential cryptanalysis against RAMus is unfeasible, thus we consider RAMus to be secure against the attacks based on differential cryptanalysis, in both the single-key and the related-key scenario.

Resistance of RAMus against linear cryptanalysis. In general, in order to distinguish a linear trail with correlation c, an adversary needs to encrypt at least c^{-2} plaintexts. According to (14; 31), the larger the size of the data sample, the more accurate the results are. In the case of RAMus, the full codebook contains up to 2^{128} (plaintext, tweak) pairs. Therefore, RAMus can be considered vulnerable against an attack based on linear cryptanalysis only if the absolute value of the correlation of its best linear trail is higher than 2^{-64}.

The correlation of the best linear trail of a cipher is computed as $c = c_S^s$, where c_S represents the best correlation associated to one active Sbox and s is the number of active Sboxes. In the particular case of RAMus, $c_S = \pm 2^{-1}$ and, according to Table 2, $s > 64$ in Scenario 2. Therefore

$$|c| < (2^{-1})^{64} \Rightarrow |c| < 2^{-64}.$$

Hence, an attack based on linear cryptanalysis against RAMus is unfeasible, thus RAMus is secure against such an attack.

6.3 The security of RAMus against integral cryptanalysis and the division property attacks

Integral cryptanalysis. Integral cryptanalysis, also known as the square attack or the saturation attack, was introduced by Knudsen in (29). An integral attack exploits the existence of an integral distinguisher defined as follows. An adversary chooses a set of plaintexts such that a set of the bits are constant, while the remaining bits (called active bits) vary through all possible values. The goal of the adversary is to find an indexing of the active bits such that the XOR sum of the corresponding ciphertexts equals to zero in some particular indexes, with probability 1. The set of plaintexts for which this property holds is called an integral distinguisher.

To design such distinguishers, the most common approach is to analyse the propagation of different properties of parts of the internal states, such as whether they are "constant" (C), "active" (A), "balanced" (B) or with the "unknow"

property (U) (i.e. a property different from the previous three ones). Note that, if components of the ciphertexts are "constant", "active" or "balanced", the XOR sum of these components results in a 0 value with probability 1. Opposed to this scenario, in the case in which components of the ciphertext have the "unknow" property, the XOR sum will be 0 with a probability less than 1.

In order to analyse the security of RAMus against integral cryptanalysis, we analyse the behaviour of the internal states in different scenarios, depending on the choice of the "active" bits. The best distinguisher identified in our analysis covers 3 rounds of RAMus. In this scenario, we consider the sets of plaintexts such that all the bits in a row have the A property. For simplicity and without loss of generality, we assume that the "active" bits are in the first row. Moreover, we impose the additional constraint that the tweak is equal to the plaintext. In this case, after the appliance of the first non-linear layer and the tweak addition, all the internal states are "constant" and each position equals to 0 (due to the cancellation between the internal state and the round tweak). After the appliance of the second non-linear layer, the state will have a "constant" value equal to $SB_C(\mathbf{0}) \oplus key$, where $\mathbf{0}$ represents the state with all 0 positions and key represents the round key. Furthermore, after the appliance of the following SB_R function, all the internal states would still have the C property.

By looking only at the tweak update function, we notice that, for the first non-linear layer, the "active" row will propagate to another "active" row, while the remaining part will be "constant". In the second round of the tweak update function, since the Sboxes are applied on a column level, each active bit will influence the properties of each corresponding column. In particular, the bits in positions 3, 4, 5, 6 will have the B property, while the remaining positions will be "constant". Therefore, the addition of the tweak in the second round would transfer the properties from the tweak to the internal state. At the end of the second round, each position of the internal state will be "balanced", a property which is also preserved through the third round. The first appliance of the non-linear layer of the 4^{th} round will determine that all the positions of the internal state will have the "unknow" property. Therefore, in this scenario, a 3-round distinguisher could be designed, as depicted in Figure 9 in the Appendix. Thus, even if the key-recovery phase of the attack could cover another 4 rounds, we consider that RAMus is resistant to integral cryptanalysis.

The division property As a new distinguishing property against block ciphers, the division property was introduced by Todo in (41) and it represents a generalization of both the integral attack and the higher-order differential cryptanalysis. In (43), the authors introduce a division property analysis technique based on the Mixed-Integer Linear Programming (MILP) problem. Since publication, this tool was used to analyse the resistance against the attacks based on the division property of several modern block ciphers, such as Princev2 (19) or GIMLI (13). By employing the same technique, we searched for the existence of integral distinguishers based on the division property for RAMus. The best distinguisher that we found covers 3.5 rounds and the data complexity

required for an attack based on this distinguisher is 2^{63} plaintexts. Note that our analysis covered both the fixed-tweak and the variable-tweak scenarios.

7 Performance

In this section we present the results of our measurements or estimates regarding the performance of the hardware implementation of RAMus and we compare them with the performance of PRINCEv2 (19), QARMA-64 (10), PRESENT (18) and SKINNY (12). We excluded from this comparison the other two block ciphers which are frequently used in RAM encryption solutions, AES and ASCON, due to the difference in their parameters' lengths.

Setup of the experiments Depending on the application, a dedicated hardware implementation is performed usually on Field-Programmable Gate Arrays (FPGA) or Application-Specific Integrated Circuits (ASIC). While ASICs are designed for a sole purpose, and the implementation is permanently drawn into silicon, the FPGAs can be reprogrammed to satisfy different purposes sequentially. Due to the versatility of the latter, all of our hardware implementations were run on the FPGA of the ZedBoard™ development kit, which uses the Xilinx Zynq®-7000 All Programable SoC (APSoC).

The first step was to identify an approach which facilitates the comparison of the hardware performance between the five targeted ciphers. In this sense, we chose to use Xilinx's Vivado High-Level Synthesis (HLS), an automated tool which transforms a high-level functional specification (such as a C or C++ implementation) into an optimized register-transfer level (RTL) descriptions, which contains the hardware implementation of the initial C/C++ code. While for RAMus we use our own C implementation, for PRINCEv2, Qarma, PRESENT and SKINNY we used the public C implementations provided by (3), (4), (2) and (5) respectively. Note that for PRINCEv2 and SKINNY we used the reference implementation provided by the authors of the ciphers.

After generating the RTL module, we used Xilinx's Vivado IP integrator to configure the hardware design, by connecting default modules (such as the Zynq architecture) with the previously generated custom one. Finally, we used Xilinx Software Development Kit (SDK), which allows for the development of embedded software applications for the hardware design formerly created.

Latency vs. throughput The performance of a hardware implementation can be evaluated with respect to several different criteria: the latency of the implementation (the speed of one encryption), its throughput (the amount of data processed in a fixed period of time), the area needed for the implementation or the power consumption.

In a sequential approach, the relation between the latency and the throughput of an implementation is given by the following formula: $N/l = t$, where N represents the amount of data to be processed, and l and t represent the

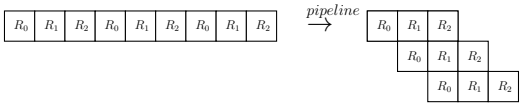

Fig. 8. The pipeline implementation of three encryptions of a 3-round block cipher. Note that the pipeline implementation is similar to a 5-round encryption.

latency and throughput, respectively. The most common approach to increase the throughput of an implementation is to use parallelization, therefore using multiple threads which perform, in parallel, the same process. This type of parallelism increases the throughput, but, in the same time, it increases the resource (CPU or area) consumption.

On the other hand, the parallelism of an FPGA, called pipelining, involves the usage of the hardware components in an optimal manner in which several instructions are overlapped during execution. The main idea behind pipelining is that a process can be divided into a set of instructions such that the output of one instruction is the input of the next one and each instruction is implemented on an independent hardware component. As soon as an instruction finishes processing an input, it is ready for the next one. Therefore, different instructions could be performed simultaneously.

For simplicity, let Enc represent a block cipher with 3 rounds, denoted R_0, R_1 and R_2. An intuitive description of the pipelining parallelism over three encryption instances can be depicted from Figure 8. In a sequential implementation the second encryption instance will start after the first one finishes. In contrast, in the case of a pipelined implementation, the output of R_0 is transmitted as an input for R_1, while R_0 can be used in parallel for the encryption of the second plaintext. Assuming that all three rounds have the same latency, performing the encryption of three plaintexts in the pipelined implementation will have the same latency as an encryption with 5 rounds in the classical implementation.

The results of our experiments In our work, we used the pipeline parallelization in two different manners. Firstly, we used the pipeline pragma provided by Vivado HLS for generating the pipelined implementation of all three targeted ciphers. We note that for the pipelined implementation of PRINCEv2 we modified the `prince_s_layer` function such that it will not be parameterized with the corresponding Sbox. Secondly, we used the pipeline pragma to estimate the throughput of the three targeted ciphers. In this sense, we measured the latency of a single pipelined round of the corresponding ciphers and we computed the total number of rounds that need to be performed for the processing of 128 encryptions (1 KB of data). Then we computed the throughput as $t = 128/(l \cdot n_r)$, where t and l represent the throughput and the latency, respectively, while n_r represent the total number of rounds.

Table 3. The performance comparison between the five targeted block ciphers. LUT, FF and BRAM stands for LookUp Table, FlipFlops and Block RAM, respectively.

	Non-pipelined			Pipelined			Throughput (KB/sec)
	Latency (μs)	Area	Power (mW)	Latency (μs)	Area	Power (mW)	
PRINCEv2	12.1	1991 LUT 2395 FF 0 BRAM	14	4.772	5730 LUT 4756 FF 0 BRAM	1705	7.29
Qarma-64	29.8	2353 LUT 3281 FF 3 BRAM	35	0.873	1050 LUT 1498 FF 0 BRAM	1682	7.95
PRESENT	25.6	1096 LUT 1320 FF 1.5 BRAM	17	9.1*	1096 LUT* 1320 FF* 1.5 BRAM*	1693*	6.75*
SKINNY	163	1210 LUT 1443 FF 2 BRAM	17	1.695	1211 LUT 1770 FF 1 BRAM	1686	7.63
RAMus	46.3	1038 LUT 1285 FF 2 BRAM	13	1.059	5065 LUT 5228 FF 0 BRAM	1669	8.00

*the PRESENT implementation contained several functions that could not be pipelined

Note that both PRINCEv2 and Qarma-64 have a self-reflection property, thus the rounds are not identical. In order to estimate the throughput for these two ciphers we individually measured all the individual rounds. We measured the middle rounds in both cases as a single round. Our estimates use the round with the highest latency, due to the fact that the processing through a lower latency round will start after the high latency one finishes.

In our experiment, we measured the latency, area and power consumption of both the pipelined and non-pipelined implementations of all five ciphers and we estimated the throughput as presented above. Table 3 presents the results of our experiments. While the results for latency, power consumption and throughput can be easily inferred from Table 3, the area of a hardware implementation is more involved. In an FPGA, a LookUp Table (LUT) stores a custom truth table which is set to simulate logic gate combinations. A flip-flop (FF) is used to store the results of LUTs, while a block RAM (BRAM) is a larger bank of RAM which is used for storing higher amounts of data inside the FPGA.

As depicted in Table 3, in the scenario of non-pipelined implementations, RAMus exhibits the lowest area and the lowest power consumption, whereas in the case of pipelined implementations, RAMus has the lower power consumption, a higher latency than PRINCEv2 and SKINNY while the area is lower than the one of PRINCEv2. Moreover, from our estimates, the throughput of RAMus is comparable with the one of Qarma-64, both being above the throughput of the other three ciphers.

Throughout performing these experiments we were surprised by the high latency of the pipelined implementation of PRINCEv2. Nonetheless, we did not find any argument which could invalidate the correctness of our experiments.

Bibliography

[1] Methodology for cryptographic rating of memory encryption schemes used in smartcards and similar devices. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_46_MEGuide_e_pdf.pdf, accessed: 2022-02-20

[2] Present C implementation. https://github.com/kurtfu/present, accessed: 2022-02-23

[3] PRINCEv2 C implementation. https://github.com/rub-hgi/princev2/tree/main/code, accessed: 2021-11-14

[4] Qarma-64 C implementation. https://github.com/Phantom1003/QARMA64, accessed: 2021-11-14

[5] Skinny C implementation. https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxza2lubnljaXBoZXJ8Z3g6NTEwY2I1MGFkZGNjMDU0MQ, accessed: 2022-02-23

[6] FIPS Publication 46-3, Data Encryption Standard (DES). https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf

[7] Albrecht, M.R., Driessen, B., Kavun, E.B., Leander, G., Paar, C., Yalçın, T.: Block Ciphers – Focus on the Linear Layer (feat. PRIDE). In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014. pp. 57–76. Springer Berlin Heidelberg (2014)

[8] AlTawy, R., Gong, G., He, M., Jha, A., Mandal, K., Nandi, M., Rohit, R.: SpoC:An Authenticated Cipher Submission to the NIST LWC Competition (2019), https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/spoc-spec-round2.pdf

[9] AlTawy, R., Gong, G., He, M., Mandal, K., Rohit, R.: Spix: An Authenticated Cipher Submission to the NIST LWC Competition (2019), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/round-1/spec-doc/spix-spec.pdf

[10] Avanzi, R.: The QARMA Block Cipher Family. Almost MDS Matrices over Rings with Zero Divisors, Nearly Symmetric Even-Mansour Constructions with Non-involutory Central Rounds, and Search Heuristics for Low-latency S-boxes. IACR Transactions on Symmetric Cryptology pp. 4–44 (2017)

[11] Azimi, S.A., Ranea, A., Salmasizadeh, M., Mohajeri, J., Aref, M.R., Rijmen, V.: A Bit-vector Differential Model for the Modular Addition by a Constant. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 385–414. Springer (2020)

[12] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) Advances

in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016)

[13] Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.X., Todo, Y., Viguier, B.: Gimli : A Cross-Platform Permutation. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. pp. 299–320. Springer International Publishing (2017)

[14] Biham, E.: On Matsui's linear cryptanalysis. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 341–355. Springer (1994)

[15] Biham, E., Anderson, R., Knudsen, L.: Serpent: A New Block Cipher Proposal. In: International Workshop on Fast Software Encryption. pp. 222–238. Springer (1998)

[16] Biham, E., Shamir, A.: Differential Cryptanalysis of DES-like Cryptosystems. Journal of CRYPTOLOGY **4**(1), 3–72 (1991)

[17] Bilgin, B., Meyer, L.D., Duval, S., Levi, I., Standaert, F.: Low AND depth and efficient inverses: a guide on s-boxes for low-latency masking. IACR Trans. Symmetric Cryptol. **2020**(1), 144–184 (2020)

[18] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-lightweight Block Cipher. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 450–466. Springer (2007)

[19] Borghoff, J., Canteaut, A., Güneysu, T., Kavun, E.B., Knezevic, M., Knudsen, L.R., Leander, G., Nikov, V., Paar, C., Rechberger, C., et al.: PRINCE–a Low-latency Block Cipher for Pervasive Computing Applications. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 208–225. Springer (2012)

[20] Boss, E., Grosso, V., Güneysu, T., Leander, G., Moradi, A., Schneider, T.: Strong 8-bit Sboxes with Efficient Masking in Hardware Extended Version. Journal of Cryptographic Engineering **7**, 1–17 (06 2017)

[21] Chabaud, F., Vaudenay, S.: Links between Differential and Linear Cryptanalysis. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 356–365. Springer (1994)

[22] Daemen, J., Govaerts, R., Vandewalle, J.: Correlation matrices. In: Preneel, B. (ed.) Fast Software Encryption. pp. 275–285. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)

[23] Daemen, J., Rijmen, V.: The Design of Rijndael. Springer-Verlag, Berlin, Heidelberg (2020)

[24] Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: ASCON v1. 2. Submission to the CAESAR Competition (2016)

[25] Grosso, V., Leurent, G., Standaert, F.X., Varıcı, K.: LS-Designs: Bitslice Encryption for Efficient Masked Software Implementations. vol. 8540 (03 2014)

[26] Gruhn, M., Müller, T.: On the Practicability of Cold Boot Attacks. In: 2013 International Conference on Availability, Reliability and Security. pp. 390–397 (2013)
[27] Joy Persial, G., Prabhu, M., Shanmugalakshmi, R.: Side Channel Attack-survey. Int J Adva Sci Res Rev **1**(4), 54–57 (2011)
[28] Kim, H., Jeon, Y., Kim, G., Kim, J., Sim, B.Y., Han, D.G., Seo, H., Kim, S., Hong, S., Sung, J., et al.: A New Method for Designing Lightweight S-boxes with High Differential and Linear Branch Numbers, and Its Application. IACR Cryptol. ePrint Arch. **2020**, 1582 (2020)
[29] Knudsen, L., Wagner, D.: Integral Cryptanalysis. In: International Workshop on Fast Software Encryption. pp. 112–127. Springer (2002)
[30] Liu, Y., Rijmen, V., Leander, G.: Nonlinear Diffusion Layers. Designs, Codes and Cryptography **86**(11), 2469–2484 (2018)
[31] Matsui, M.: Linear Cryptanalysis Method for DES Cipher. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 386–397. Springer (1993)
[32] Meyer, L.D., Varici, K.: More Constructions for Strong 8-bit S-boxes with Efficient Masking in Hardware (2017)
[33] Mourouzis, T.: Optimizations in Algebraic and Differential Cryptanalysis. Ph.D. thesis, UCL (University College London) (2015)
[34] Nikova, S., Rechberger, C., Rijmen, V.: Threshold Implementations Against Side-channel Attacks and Glitches. In: International Conference on Information and Communications Security. pp. 529–545. Springer (2006)
[35] Ruisanchez, C.P.: A New Algorithm to Construct S-boxes with High Diffusion. International Journal of Soft Computing, Mathematics and Control (IJSCMC) **4**(3) (2015)
[36] Saarinen, M.J.O.: Cryptographic Analysis of All 4× 4-bit S-boxes. In: International Workshop on Selected Areas in Cryptography. pp. 118–133. Springer (2011)
[37] Sarkar, S., Mandal, K., Saha, D.: On the Relationship Between Resilient Boolean Functions and Linear Branch Number of S-Boxes. In: International Conference on Cryptology in India. pp. 361–374. Springer (2019)
[38] Sarkar, S., Syed, H.: Bounds on Differential and Linear Branch Number of Permutations. In: Australasian Conference on Information Security and Privacy. pp. 207–224. Springer (2018)
[39] Stern, J., Vaudenay, S.: Cs-cipher. In: Vaudenay, S. (ed.) Fast Software Encryption, 5th International Workshop, FSE '98, Paris, France, March 23-25, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1372, pp. 189–205. Springer (1998)
[40] Stewin, P., Bystrov, I.: Understanding DMA Malware, isbn = 978-3-642-37299-5 (07 2012)
[41] Todo, Y.: Structural Evaluation by Generalized Integral Property. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 287–314. Springer (2015)
[42] Vaudenay, S.: On the security of cs-cipher. In: Knudsen, L.R. (ed.) Fast Software Encryption, 6th International Workshop, FSE '99, Rome, Italy,

March 24-26, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1636, pp. 260–274. Springer (1999)

[43] Xiang, Z., Zhang, W., Bao, Z., Lin, D.: Applying MILP Method to Searching Integral Distinguishers Based on Division Property for 6 Lightweight Block Ciphers. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016. pp. 648–678. Springer Berlin Heidelberg (2016)

Appendix 1. The byte description of the Sbox S

Table 4. The Sbox S. The output associated to the hexadecimal input xy can be depicted from the intersection of the row $x0$ and the column $0y$. For example, $S(c2) = 5d$.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	33	24	4e	c1	8d	9a	e7	15	f0	7f	59	ab	02	68	bc	d6
10	42	55	3f	b0	fc	eb	96	64	81	0e	28	da	73	19	cd	a7
20	e4	f3	99	16	5a	4d	30	c2	27	a8	8e	7c	d5	bf	6b	01
30	1c	0b	61	ee	a2	b5	c8	3a	df	50	76	84	2d	47	93	f9
40	d8	cf	a5	2a	66	71	0c	fe	1b	94	b2	40	e9	83	57	3d
50	a9	be	d4	5b	17	00	7d	8f	6a	e5	c3	31	98	f2	26	4c
60	7e	69	03	8c	c0	d7	aa	58	bd	32	14	e6	4f	25	f1	9b
70	51	46	2c	a3	ef	f8	85	77	92	1d	3b	c9	60	0a	de	b4
80	0f	18	72	fd	b1	a6	db	29	cc	43	65	97	3e	54	80	ea
90	f7	e0	8a	05	49	5e	23	d1	34	bb	9d	6f	c6	ac	78	12
a0	95	82	e8	67	2b	3c	41	b3	56	d9	ff	0d	a4	ce	1a	70
b0	ba	ad	c7	48	04	13	6e	9c	79	f6	d0	22	8b	e1	35	5f
c0	20	37	5d	d2	9e	89	f4	06	e3	6c	4a	b8	11	7b	af	c5
d0	86	91	fb	74	38	2f	52	a0	45	ca	ec	1e	b7	dd	09	63
e0	cb	dc	b6	39	75	62	1f	ed	08	87	a1	53	fa	90	44	2e
f0	6d	7a	10	9f	d3	c4	b9	4b	ae	21	07	f5	5c	36	e2	88

Appendix 2. The integral distinguisher described in Section 6.3

We recall that, for this distinguisher, the first row is "active", with the additional constraint that the tweak is equal to the plaintext.

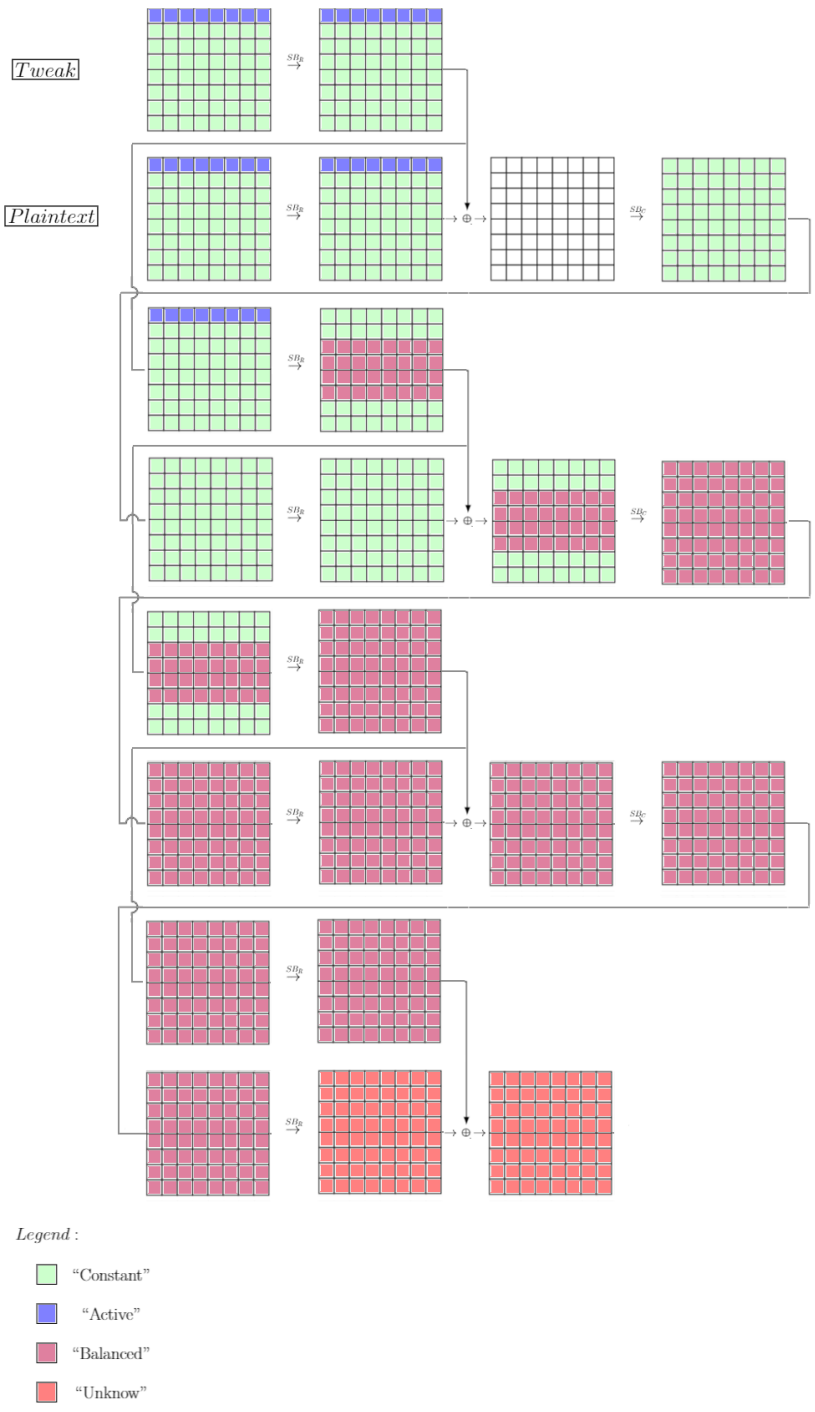

Fig. 9. The 3-round integral distinguisher described in Section 6.3.