MDPI

Article

Measuring Performances of a White-box Approach in the IoT Context

Daniele Giacomo Vittorio Albricci [1], Michela Ceria [2], Federico Cioschi [3], Nicolò Fornari [4], Arvin Shakiba [5] and Andrea Visconti [6,*]

[1] Department of Computer Science, Università degli Studi di Milano, via Celoria 18, 20133, Milan, Italy; daniele.albricci@studenti.unimi.it
[2] Department of Computer Science, Università degli Studi di Milano, via Celoria 18, 20133, Milan, Italy; michela.ceria@gmail.com
[3] Department of Computer Science, Università degli Studi di Milano, via Celoria 18, 20133, Milan, Italy; federico.cioschi@studenti.unimi.it
[4] Open Systems AG, Räffelstrasse 29, 8045 Zurich, Switzerland; nfornari@open-systems.com
[5] Department of Computer Science, Università degli Studi di Milano, via Celoria 18, 20133, Milan, Italy; arvin.shakiba@studenti.unimi.it
[6] Department of Computer Science, Università degli Studi di Milano, via Celoria 18, 20133, Milan, Italy; andrea.visconti@unimi.it
* Correspondence: andrea.visconti@unimi.it (A.V.)

Version September 2, 2019 submitted to Symmetry

Abstract: Internet of Things refers to all the smart objects that are connected to other objects, devices or servers and that are able to collect and share data, in order to "learn" and improve their functionalities. Smart objects suffer from lack of memory and computational power, since they are usually lightweight. Moreover, their security is weakened by the fact that smart objects can be placed in unprotected environments, where adversaries are able to play with the symmetric-key algorithm used and the device on which the cryptographic operations are executed. In this paper, we focus on a family of white-box symmetric ciphers SPNbox, extending and improving our previous paper on the topic presented at WIDECOM2019. We highlight the importance of white-box cryptography in the IoT context, but also the need to have a fast black-box implementation (server-side) of the cipher. We show that, modifying an internal layer of SPNbox, we are able to increase the key length and to improve the performance of the implementation. We measure these improvements (a) on 32/64-bit architectures and (b) in the IoT context by encrypting/decrypting 10,000 payloads of lightweight messaging protocol MQTT.

Keywords: symmetric cryptography; IoT; MQTT; white-box approach; the SPNbox family

1. Introduction

The name *Internet of Things* (IoT), coined by the MIT researcher Kevin Ashton [1], usually refers to *smart objects*, connected through the Internet to other sensors, devices and servers with which collect and/or share data for improving their functionalities. IoT can also be combined with other technologies, for example with cloud computing [2]. It is possible to create a sustainable smart home aiming to reduce resources' consumption or develop specific applications in the medical field [3] such as wearable devices which monitor our physical conditions, specific devices used to check patients with chronic illnesses, and so on. Data collected by IoT devices need to be (a) processed to form informations by applying, for example, data mining techniques [4]; (b) evaluated in order to make decision by adopting agent based models [5,6], bayesian decision models [7], fuzzy logic [8] and so on; (c) protected from attacks, failures and leaks during communication [2].

Several issues have to be faced in securing IoT applications. An important example is given by the intrinsic constraints of the devices [9], that usually have a small amount of memory and cannot perform heavy computations. It is very likely to have such devices in non-protected environment, where an adversary can access them and perform attacks. In particular, she/he can perform an analysis of the controlled binary [10] or perform differential fault analysis [11,12]. Moreover, since these devices are connected, compromising one of them can open the way to botnet attacks [2,3]. We can observe that we are exactly in a white-box framework, and white-box cryptography [13] has been first developed to cope with the scenario in which an attacker can physically interact both with the implementation of the used cryptographic algorithm and with the device on which the encryption/decryption operations are executed. The usually studied scenario, namely black-box, in which the execution cannot be observed nor modified by the attacker, is not always suitable for IoT applications [14]. The reader can think about what happens in the context of digital rights management where discovering the key means to have the possibility to spread digital contents to people that have not payed such contents.

The effort of researchers towards white-box cryptographic schemes materialized with [15] and [13] where white-box versions of AES and DES have been implemented. Nevertheless, it is important to remark that these implementations have been attacked via algebraic attacks [16] (improved by [17]) ,[18],[19], [20]. Moreover, also [21] can easily break Chow's implementations.

The need to have white-box algorithms for practical applications leads to develop some specific algorithms. Examples of block ciphers developed to be employed in the framework of white-box cryptography are ASASA [22] and SPACE [23]. However, these ciphers are not free of drawbacks or weaknesses. In particular, decomposition attacks can affect ASASA's security while SPACE is heavy from a computational point of view [24]. An important step forward for white-box cryptography, was the development of SPNbox [24], another block cipher that relies on internal block ciphers with the aim to reduce the computation time. In [9], the problem of intrinsic constraints on computational power and memory of IoT devices in unprotected environment is addressed. The authors refer to smart objects with limited computational power and memory that may contain sensitive data and can be easily lost or stolen. Differently from AES/DES white-box implementations, the authors do not decline a well-known cipher into the new framework, but they develop a new one, relying on a modification of Lai-Massey structure. The crucial point is that only the encryption is thought to be done on the IoT constrained device, while the decryption phase is supposed to be done on a computer or server and in a black box scenario. In [25] the authors refer to embedded distributed devices which collect and securely send information to centralized servers. Subsequently, these servers decrypt and process all the information. As previously mentioned, the collected information may be sensitive and it is possible for an attacker to get control of the whole device. The scheme proposed in [25] is lightweight and suitable for constrained devices. In particular, such a new design has the following peculiarities:

- the employed operations are very simple; they essentially consist of lookup tables and bit operations;
- the lookup tables and the structure containing sensitive data are small in memory;
- the provided security is medium-level ($\sim 2^{63}$) and protection is ensured for reasonable amount of time;
- it is possible to update the key at small costs.

The scheme is based on a Fesitel structure, but it adds two bijections, as a defence against attacks. Moreover, to cope with structural cryptanalysis [25,26] different size components are used.

This paper improves of a previous work entiteled "White-box Cryptography: A Time-security Trade-off for the SPNbox Family" [27], presented by F.Cioschi, N.Fornari and A.Visconti at the 2nd International Conference on International Conference on Wireless, Intelligent and Distributed Environment for Communication (WIDECOM 2019). In this paper, we (a) introduce the white-box approach in the IoT context, explaining the importance of protecting data in an environment where attackers have full control over the whole system; (b) explain the importance of having a fast black-box implementation of a white-box cipher; (c) summarize our previous idea [27] explaining how to modify

the internal block ciphers of the SPNbox family in order to increase the size of the key space; (d) measure the performance of a black-box implementation (server-side) on 32- and 64-bit architectures and by encrypting/decrypting 10,000 payloads of a lightweight messaging protocol — i.e., MQTT — which contains the data sent over the Internet.

The remainder of the paper is organized as following. In Section 2, block ciphers are introduced. In Section 3, we present several white-box implementations and related attacks published in literature. In Section 4 an 5, we summarize two block ciphers' families, namely, SPACE and SPNbox, which are white-box friendly by design. In Section 6, we explain the importance of increasing the number of bits of the key used in each round. In Section 7, the testing activities are presented. Finally, Section 8 is devoted to discussion and conclusions.

2. Block ciphers

There exist two main families of block ciphers: substitution-permutation network (SPN) and Feistel network. The main difference between them is that Feistel networks play only with one half of the cipher state in each round.

2.1. Substitution-Permutation Networks

A substitution-permutation network (SPN) is a design for block ciphers proposed by Shannon in [28], where he suggested to use multiple mixing layers interleaving substitutions and permutations. Although weak on its own, applying substitutions and then permutations presents good "mixing" properties. Substitutions contribute to local *confusion* and permutations spread such a local confusion to the more distant subblocks, thus providing *diffusion* [28]. If a single input bit is flipped, it affects the m output bits of a specific S-box which, subsequently, are sent to different S-boxes by a permutation. Considering the output of such a network, about fifty percent of the bits are affected by this change. Therefore an outcome of a single bit change at the input is difficult to predict, especially if the bit of the secret key are XORed into the block between the encryption layers. In order to get better diffusion properties, several block ciphers adopt linear (as in the case of AES) or affine mappings instead of permutations.

Definition 1. *Let $\phi : (\mathbb{F}_2)^r \times (\mathbb{F}_2)^l \to (\mathbb{F}_2)^r$, with $r = bt$ be a block cipher with N rounds. Let $k \in (\mathbb{F}_2)^l$ be the cipher key and $(k^{(0)}, \cdots, k^{(N)})$ be the $N+1$ round keys generated by k through the key schedule. Then ϕ is an SPN block cipher if*

$$\phi_k(x) = \tau_N \circ \tau_{N-1} \circ \ldots \circ \tau_0(x) \qquad x \in (\mathbb{F}_2)^r$$

where $\tau_i = \sigma_{k^{(i)}} \circ \lambda^{(i)} \circ \gamma^{(i)}$ and

- *$\gamma^{(i)} : (\mathbb{F}_{2^t})^b \to (\mathbb{F}_{2^t})^b$ is a non linear substitution*
- *$\lambda^{(i)} \in AGL((\mathbb{F}_2)^r)$ where $AGL((\mathbb{F}_2)^r)$ is the subgroup of the affine transformations of $(\mathbb{F}_2)^r$*
- *$\sigma_{k^{(i)}}$ is the addition with the round key*

$$\begin{array}{rccc} \sigma_{k^{(i)}} : & (\mathbb{F}_2)^r & \to & (\mathbb{F}_2)^r \\ & x & \mapsto & x \oplus k^{(i)} \end{array}$$

where by $\oplus$ we denote the bitwise addition (XOR)

2.2. Feistel networks

A Feistel network is a block cipher introduced by H. Feistel and D. Coppersmith in 1973 [29] which has the advantage of having the encryption and decryption functions almost identical, making the implementation easier and cheaper compared to translation based ciphers.
Let us define the following functions before describing the encryption process.

Definition 2. *Let π_t be a projection, with $t \in \{1, \ldots, 2n\}$, defined as:*

$$\begin{array}{rccc} \pi_t : & \mathbb{F}^{2n} & \to & \mathbb{F}^t \\ & (x_1, \ldots, x_{2n}) & \mapsto & (x_1, \ldots, x_t) \end{array}$$

Considering $x \in \mathbb{F}^{2n}$ as a vector of bits, we can use projection π_t to choose the t *most* significant bits of x.

Definition 3. *Let ϱ_t be a projection, with $t \in \{1, \ldots, 2n\}$, defined as:*

$$\begin{array}{rccc} \varrho_t : & \mathbb{F}^{2n} & \to & \mathbb{F}^t \\ & (x_1, \ldots, x_{2n}) & \mapsto & (x_{2n-t+1}, \ldots, x_{2n}) \end{array}$$

In the same way, using projection ϱ_t we can choose the t *least* significant bits of x.

Given $m \in \mathbb{F}^{2n}$ (a message) and $k \in \mathbb{F}^l$ (a secret key), for some positive integer l, the encryption process works as follows:

1. $N + 1$ round keys $k_0, ..k_N$ are generated from k by means of the key schedule
2. message m is split into a left block and right block, initialized as

$$L_0 = \pi_n(m) \qquad R_0 = \varrho_n(m)$$

3. for $i \in \{1, \ldots, N + 1\}$ the round function is applied in the following way:

$$L_i = R_{i-1} \qquad R_i = L_{i-1} \oplus F(R_{i-1}, k_{i-1}) \tag{1}$$

4. final ciphertext c is (R_{N+1}, L_{N+1}).

Encryption and decryption only differ in the reverse order of the round keys, as a matter of fact the decryption of ciphertext (R_{N+1}, L_{N+1}) is accomplished computing for $i \in \{N, N-1, \ldots, 0\}$

$$R_i = L_{i+1} \qquad L_i = R_{i+1} \oplus F(L_{i+1}, k_i) \tag{2}$$

An advantage of Feistel networks is that Feistel function F is non-necessarily invertible.This can be clearly seen by analyzing how encryption and decryption work (see Equations 1 and 2).

3. The White-Box approach

The White-Box approach aims to avoid key recovery attacks by embedding the cryptographic key into a robust representation of the cipher. Consider a block cipher ϕ. We compute a map $\psi : \mathbb{F}^n \to \mathbb{F}^n$ such that, given a key $\bar{k} \in \mathbb{F}^l$, it holds $\phi(x, \bar{k}) = \psi(x) \ \ \forall x \in \mathbb{F}^n$. If an attacker knows even both ϕ and ψ, it should be very hard for him to find out the key.

Example 1. *Let ϕ and ψ be defined as follows:*

$$\begin{array}{lll} \phi(x) := k + x \mod 4 & x \in \{0, \ldots, 3\} \\ \psi(x) := S[x] & S = [3, 0, 1, 2] \end{array}$$

If $k = 3$, we can consider ψ as a white-box implementation of ϕ, by representing ψ as a lookup table.

The first white-box AES implementation has been proposed by Chow et al. in [13]. The authors suggest that key extraction can be avoided by a careful use of lookup tables. In particular, given a secret key and a block cipher, it is possible to create a lookup table which maps the plaintext in a corresponding ciphertext. In some cases, this lookup table may be huge and unusable due to its

dimension. Therefore, a block cipher ϕ can be represented as a network of smaller lookup tables (see Figure 1) that have to be read in a particular order [13]. Unfortunately, in the white-box framework an adversary has full access to these tables, exposing the cipher to possible attacks. Since there is no reason to make an attacker's life easier, tables can be protected by means of internal encodings [13]. This means that a map is composed after table i and its inverse before table $(i+1)$, leaving the ciphertext unchanged. However, internal encoding does not protect against code-lifting attacks. Indeed, an attacker may recover the tables of the cipher and understand their concatenation order. Doing so, she/he is able to decrypt messages even though he had not recovered the secret key. Therefore, another protection is required: external encoding. Internal and external encodings are also discussed in [30], while a different approach, based on polynomial algebra techniques [31], gave rise to a perturbated white-box implementation of AES [32], broken by [33] in 2010.

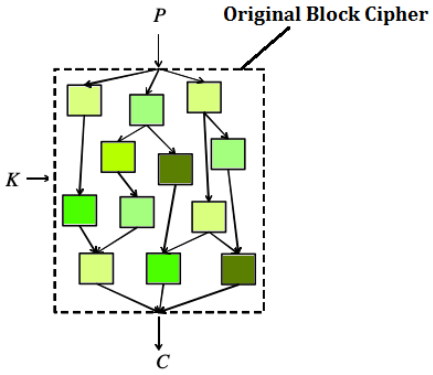

Figure 1. Table-based white-box implementation: the key k is scrambled by a network of lookup tables

Chow's work is a milestone for white-box cryptography and its framework has also been used by some subsequent works such as [34] and [35]. However, researchers found attacks also for these new approaches:

- White-Box AES Implementation: Chow [13]; Attack: [16]; Work Factor: 2^{30};
- White-Box AES Implementation: Karroumi [34]; Attack: [18] and [20]; Work Factor: 2^{22};
- White-Box AES Implementation: Xiao Lai [35]; Attack: [18]; Work Factor: 2^{32};
- White-Box AES Implementation: Xiao Lai [35] generic linear version; Attack: [18]; Work Factor: 2^{38};
- White-Box AES Implementation: Xiao Lai [35] affine/non-affine version; Attack: [19]; Work Factor: at least 2^{49}.

The attacks listed above may require to know the internal data representation and sometimes this means to produce a significant reverse engineering effort. An improved AES implementation is given in [36]. This implementation is immune to attacks described in [16,18] but it is not to the one presented in [37].

The first paper aiming to break all white-box implementations belonging to the framework introduced in [13] is [19], but it has the weak point to require some additional hypotheses. Differently, [38] breaks all the papers in Chow's framework by solving the affine equivalence problem (see [39] and [40]). Chow's framework has also been used by [41] and subsequently attacked by [42].

A significant advance from the attacker's point of view became feasible by shifting the focus from the attacks previously described to side channel attacks [43]. In particular, new approaches to verify the security of a white-box implementation have been proposed in [44] where Bos et al. present

differential fault analysis (DFA) and differential computational analysis (DCA) attacks[1]. In addition, [47] and [48] explained more formally why DCA is effective against linear and nibble encoding, [43] provides an extensive analysis on the effectiveness of DCA, and [49] gives a general protection method for white-box implementations against DCA.

Obfuscation techniques or the randomization of the location of the lookup tables can be used to enhance security of white-box algorithms [50], while [51] examines how these techniques are successful against both DCA and differential power attacks (DPA). The paper [52] exploits noncommutative groups to obfuscate operation that should be made on commutative ones and it is employed in the IoT framework. Finally, an evaluation on software protections to white-box implementations is provided by [51] .

Some improvements to DCA have been developed by [53] and [43]. The first one extends DCA to successfully address implementations using masking and shuffling techniques [53]. The second one provide a DCA-like collision attack with a good complexity [43].

Some paper such as [54], [55] and [56] address the problem of incompressibility or code hardness. The idea is that an attacker in the white-box framework should not be able to rewrite the code of some implementation in order to decrease the code-hardness. In [54] two incompressible white-box schemes called "WhiteKey" and "WhiteBlock" are introduced and one instance for each scheme is provided (called PuppyCipher and CoureurDesBois respectively), [55] describes the concept of code-hardness, time-hardness and memory-hardness, while [56] provides a new incompressible white-box implementation based on the assumption of one-way permutations.

We conclude our extensive analisys of implementations and attacks, citing a white-box signature scheme [57] and the methods [58] used to attack the most resistant implementation submitted to the white-box competition called "CHES 2017 CTF Challenge".

In the sequel, we will analyze in detail two family of white-box cipher called SPACE (Section 4) and SPNbox (Section 5).

4. SPACE: a block cipher

SPACE is a block cipher developed by Bogdanov and Isobe in [23], that is based on a Feistel network. This cipher is designed so that security against key extraction in the white-box context reduces to the well studied problem of key recovery for block ciphers in the standard black-box setting.

SPACE is a generalized Feistel network [29]. Given a message $m \in \mathbb{F}^n$ and a secret key $k \in \mathcal{K}$, it encrypts m to a ciphertext $c \in \mathbb{F}^n$. In describing SPACE, three quantities are often employed: $n, n_a, n_b \in \mathbb{N}$. In particular, in [23] $n = 128$, $n_a \in \{8, 16, 24, 32\}$ and $n_b = n - n_a$.

We summarize here the encryption procedure:

1. The state X^r at round r can be seen as given by $l = n/n_a$ vectors $x_i^r \in \mathbb{F}^{n_a}$ so
$$X^r = \{x_0^r, x_1^r, \ldots, x_{l-1}^r\}.$$
2. $X^0 = m$, so it is initialized with the plaintext.
3. For $r \in \{1, \ldots, R+1\}$ the state is updated this way:
$$X^{r+1} = \left(F_{n_a}^r(x_0^r) \oplus (x_1^r || x_2^r || \ldots || x_{l-1}^r)\right) || x_0^r$$
where $F_{n_a}^r : \mathbb{F}^{n_a} \to \mathbb{F}^{n_b}$ is the Feistel function and $||$ is the concatenation.
4. $X^{R+1} = c$ so we have found the ciphertext.

[1] Further information on fault-injection and differential power analysis attacks can be found in [45] and [46] respectively.

At each encryption round, the Feistel function takes x_0^r as input. Then $F_{n_a}^r(x_0^r)$ is added to the rest of the state $(x_1^r||\ldots||x_{l-1}^r)$. The first n_b bits of the new state are given by the result of this operation. The last n_a are filled with x_0^r.

Now, consider π_t be the projection of Definition 2 and the Feistel function $F_{n_a}^r$ used by SPACE, specified in Definition 4.

Definition 4. *Let ϕ_k be a block cipher and r the round number represented in binary with n_b digits (so we see it as an element of $\mathbb{F}^{n_b}$). The* Feistel function $F_{n_a}^r$ *is defined as*

$$\begin{array}{rccl} F_{n_a}^r(x): & \mathbb{F}^{n_a} & \to & \mathbb{F}^{n_b} \\ & x & \mapsto & (\pi_{n_b}(\phi_k(\overbrace{0,\ldots,0}^{n_b}||x))) \oplus r. \end{array}$$

We give a specific notation for the round independent part of F_{n_a}.

Definition 5. *The round independent part of the Feistel function F_{n_a} is*

$$\begin{array}{rccl} F'_{n_a}: & \mathbb{F}^{n_a} & \to & \mathbb{F}^{n_b} \\ & x & \mapsto & \pi_{n_b}(\phi_k(\overbrace{0,\ldots,0}^{n_b}||x)) \end{array}$$

Notice that, differently from traditional Feistel networks, SPACE does not use round keys. There is one secret key k used by ϕ_k. This secret key cannot be hardcoded, hence $\mathbb{F}'_{n_a}$ is implemented as a look-up table. The reader might ask himself the reason for designing SPACE over another block cipher ϕ_k when ϕ_k could be directly implemented as a look-up table. It turns out that this second possibility cannot be developed. If we were to implement ϕ_k as a look-up table we would need $2^n \cdot n$ bits of space:

$$\begin{array}{ccc} \overbrace{(0,\ldots 0,0)}^{n} & \mapsto & \overbrace{\phi_k(0,\ldots 0,0)}^{n} \\ (0,\ldots 0,1) & \mapsto & \phi_k(0,\ldots 0,1) \\ & \vdots & \\ (1,\ldots 1,1) & \mapsto & \phi_k(1,\ldots 1,1) \end{array}$$

For $n = 128$ the construction of such a look-up table is practically impossible. Therefore Bogdanov and Isobe propose to truncate the output of ϕ_k, computed over a smaller domain:

$$\begin{array}{ccc} (\overbrace{0,\ldots,0}^{n_b}||\overbrace{0,\ldots 0,0}^{n_a}) & \mapsto & \overbrace{\pi_{n_b}(\phi_k(0,\ldots,0||0,\ldots 0,0)}^{n_b} \\ (0,\ldots,0||0,\ldots 0,1) & \mapsto & \pi_{n_b}(\phi_k(0,\ldots,0||0,\ldots 0,1)) \\ & \vdots & \\ (0,\ldots,0||1,\ldots 1,1) & \mapsto & \pi_{n_b}(\phi_k(0,\ldots,0||1,\ldots 1,1)) \end{array}$$

Figure 2. The value of each image of $F'_{n_a}(x)$ is saved as a row in a look-up table. Every row is indexed by the value of x, $x \in \{0,\ldots,2^{n_a}-1\}$.

Since the first n_b zeros are used as padding in order to form an n-bit input to provide to ϕ_k, it is completely useless to store them, hence the look-up table implementation needs $2^{n_a} \cdot n_b$ bits. Thus, the size of the tables for different values of $n_a \in \{8, 16, 24, 32\}$ — SPACE(n_a,R), where R is the suggested number of rounds — is the following:

- SPACE-(8,300); Table: 3.84 KB
- SPACE-(16,128); Table: 918 KB
- SPACE-(24,128); Table: 218 MB

- SPACE-(32,128); Table: 51.5 GB

Notice that (1) AES white-box implementations of Chow et al. [13] and Xiao Lai [35] has a table of 752 KB and 20.5 MB respectively; (2) not all n_a values are suitable, indeed, for $n_a = 32$ and $n_a = 24$ the size of the table is not good enough to be used in practice. On the contrary, for $n_a = 16$ the table has the same size of that described in [13].

5. The SPNbox Family

The SPACE family of space-hard block ciphers [23] benefits of the Feistel structure from a security point of view and prevents the use of parallel execution (see Section 4). However, as suggested in [24], using an SPN-type design it is possible to satisfy the requirement of parallelism maintaining a suitably high level of space hardness. Thus, Bogdanov et al. described the SPNbox family of space-hard block ciphers [24]. Let us briefly explain their idea.

SPNbox-n_{in} is a substitution-permutation network (SPN) with a block length of n bits, a k-bit secret key, and based on n_{in}-bit substitution boxes.

State:

The state of SPNbox-n_{in} is representable as a vector of $t = n/n_{in}$ elements of n_{in} bits each:

$$X = \{X_0, ..., X_{t-1}\}$$

Key Schedule:

The k-bit master key is expanded, $k_0, ..., k_{R_{n_{in}}}$ round keys of n_{in} bits, by means of a Key Derivation Function (KDF) — e.g., PBKDF2 [59–62], ARGON2 [63], Scrypt [64], and so on:

$$(k_0, ..., k_{R_{n_{in}}}) = KDF(k, n_{in} \cdot (R_{n_{in}} + 1))$$

Round Transformation:

We encrypt a plaintext X^0 and we get a ciphertext X^R, by using the following R transformations — e.g., $R = 10$:

$$X^R = (\bigcirc_{r=1}^{R}(\sigma^r \circ \theta \circ \gamma))(X^0)$$

The nonlinear layer γ is a substitution layer where t identical bijective n_{in}-bit S-boxes depending on the key are applied to the state:

$$\begin{aligned} &\gamma : \mathbb{F}(2^{n_{in}})^t \to \mathbb{F}(2^{n_{in}})^t \\ &(X_0, ..., X_{t-1}) \mapsto (S_{n_{in}}(X_0), ..., S_{n_{in}}(X_{t-1})) \end{aligned} \tag{3}$$

These identical S-boxes are constituted by an internal small block cipher of block length n_{in} bit.

The linear layer θ, a diffusion layer, applies a $t \times t$ MDS matrix to the state:

$$\theta : \mathbb{F}(2^{n_{in}})^t \to \mathbb{F}(2^{n_{in}})^t$$

$$(X_0, ..., X_{t-1}) \mapsto (X_0, ..., X_{t-1}) \cdot M_{n_{in}}$$

The affine layer σ^r takes the state and adds round-dependent constants to it:

$$\sigma^r : \mathbb{F}(2^{n_{in}})^t \to \mathbb{F}(2^{n_{in}})^t$$

$$(X_0, ..., X_{t-1}) \mapsto (X_0 \oplus C_0^r, ..., X_{t-1} \oplus C_{t-1}^r),$$

with $C_i^r = (r-1) \cdot t + i + 1$ for $0 \leq i \leq t-1$.

The Underlying Small Block Ciphers:

The identical n_{in}-bit S-boxes in the γ layer (which depend on the key) are block ciphers. They are based on the round transformation of AES and they are formed by $R_{n_{in}}$ rounds operating on a state $x = \{x_0, ..., x_{l-1}\}$ of l bytes, where $l = n_{in}/8$:

$$S_{n_{in}} : \mathbb{F}(2^8)^l \to \mathbb{F}(2^8)^l$$

$$x \mapsto (\bigcirc_{i=1}^{R_{n_{in}}} (AK^i \circ MC_{n_{in}} \circ SB))(AK^0(x))$$

where SB, MC and AK indicate the AES transformations SubBytes, MixColumns and AddRoundKey, respectively. Notice that (a) the number of rounds $R_{n_{in}}$ that [24] suggests are R_{32}=16, R_{24}=20, R_{16}=32 and R_8=64; (b) different matrices are employed in the $MC_{n_{in}}$ round transformation. More precisely, for n_{in}=32 we use the MC matrix of AES, while in the other cases a sub-matrix of MC is used. If n_{in}=8, $MC_{n_{in}}$ is the identity map's matrix. Note that, as for the Feistel function in SPACE, in the white-box setting the small block ciphers $S_{n_{in}}$ are implemented as lookup tables.

6. Issues and possible solutions

Although the white-box implementation of the cipher is very important, it may have some limitations due to the key embedded into the device. If several devices have to communicate with a server and such devices do not support TLS due to insufficient resources, the server needs to manage a number of keys (pre-shared or not) in order to decrypt the messages. In a white-box context this means having a number of different implementations that run on our server and this is not a good idea. Therefore, the server will be provided with a fast black-box implementation of the cipher.

Figure 3 helps us to visualize this idea, where a white-box implementation runs on a number of devices and a fast black-box implementation runs on our server.

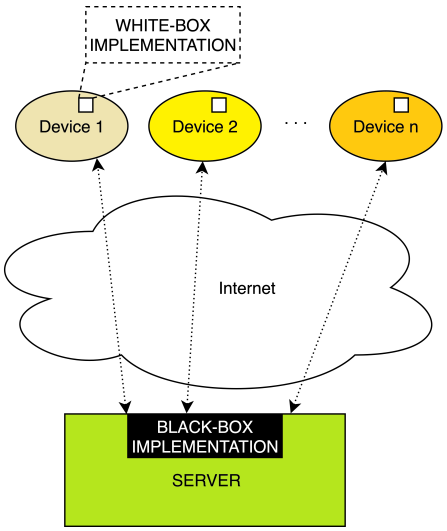

Figure 3. A black-box implementation (server-side)

In order to design a fast black-box implementation of a white-box cipher, we modify the inner round described in Section 5, increasing the number of bits of the key used in each round. In particular, we replace the AES' ShiftRow transformation, omitted by [24], with a key-dependent circular bit shift transformation (see Figures 4 and 5).

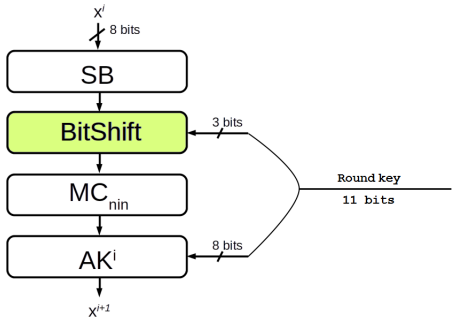

Figure 4. A BitShift key-dependent: Increasing the size of the round key from 8 to 11 bits.

If we are shifting 8 bits of the state, i.e., n_{in}=8, 3 bits are required to execute the circular shift. Thus, we use 11 bits of the key in each round i: 8 of them for the AK^i transformation and 3 for the BitShift transformation. If the state doubled, tripled, or quadrupled, i.e., n_{in}=16, 24, 32, the bits of the key used are $11 \times 2 = 22$, $11 \times 3 = 33$ and $11 \times 4 = 44$ respectively.

Notice that the implementation of [24] employs the AES-NI instructions, while the idea described in this paper does not. In the encryption phase (n_{in}=32, 24, 16), the matrices involved in the computation of the MixColumns transformation (A_{24}, and A_{16} for short) are sub-matrices of that used in AES (A_{32}). On the contrary, in the decryption phase, we need to invert A_{24} and A_{16}. Since their inverse matrices are not sub-matrices of A_{32}^{-1} and the decryption instruction of AES-NI is based only on A_{32}^{-1}, for n_{in}=24,16 we cannot use the AES-NI instructions. Anyway, in IoT context, the impossibility of using AES-NI instructions is not a problem in itself because not all IoT devices support this instruction set.

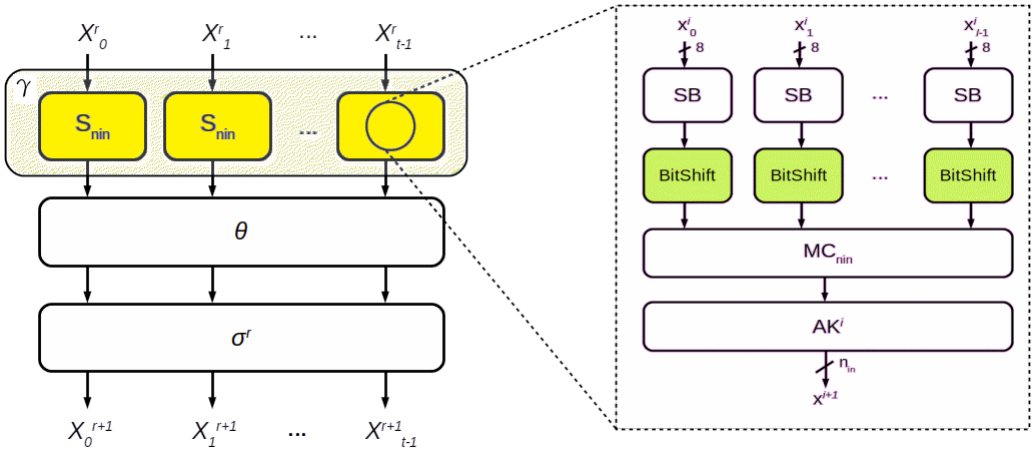

Figure 5. SPNbox: a new inner round γ

7. Testing activities

The testing activity reported is twofold. In the first part, we measure the performance of internal layer γ (see Algorithm 1) — the external part (layer θ and σ) is exactly the same as in [24], so it would be pointless to evaluate it.

In the second part, as explained in Section 6, we examine the cipher in the IoT context, where black-box and white-box implementations are involved.

7.1. 32/64-bit architectures

We compared the performance of internal layer γ (yellow rectangles of Figure 5) with and without BitShift transformation (green rectangles of Figure 5) for different n_{in} sizes. We avoid the operations involved in θ and σ layers.

Algorithm 1: Layer γ with BitShift transformation

```
Function SPNbox(state):
    for r = 1 → R do
        layer_gamma(state)
        layer_theta(state)
        layer_sigma(state, r)
Function layer_gamma(state):
    Set n_in = 8, 16, or 32
    foreach chunk of length n_in in state do
        S_{n_in}(state)
Function S_{n_in}(state):
    AddRoundKey(state)
    for i = 1 → R_{n_in} do
        SubBytes(state)
        BitShift(state)
        MixColumns(state)
        AddRoundKey(state)
Function layer_theta(state):
    return
Function layer_sigma(state,r):
    return
```

We run our code on laptops with different hardware configurations. More precisely, our laptops are equipped with

- Intel® Core™ i3-330M, 2.13 GHz processor with 3 MB SmartCache, 8 GB RAM and Ubuntu 18.04.1 LTS 64-bit. The source code has been compiled with GCC 7.3.0 with `-O3` optimization enabled (see Table 1);
- Intel® Core™ i3-350M, 2.26 GHz processor with 3 MB SmartCache, 8 GB RAM and Ubuntu 18.04.2 LTS 64-bit. The source code has been compiled with GCC 7.4.0 with `-O3` optimization enabled (see Table 2);
- Intel® Core™ i7-2860QM, 2.50/3.60 GHz processor with 8 MB SmartCache, 16 GB RAM and Kubuntu 18.10 64-bit. The source code has been compiled with GCC 7.3.0 with `-O3` optimization enabled (see Table 3);
- Intel® Core™ i7-5500U, 2.40/3.00 GHz processor with 4 MB Cache, 8 GB RAM and Ubuntu 18.04.2 LTS 64-bit. The source code has been compiled with GCC 7.4.0 with `-O3` optimization enabled (see Table 4);
- Intel® Core™ i7-8550U CPU, 1.80/4.00 GHz processor with 8 MB SmartCache, 32 GB RAM and Ubuntu 18.04.2 LTS 64-bit. The source code has been compiled with GCC 7.4.0 with `-O3` optimization enabled (see Table 5);
- Intel® Core™ i3-350M, 2.26 GHz processor with 3 MB SmartCache, 4 GB RAM and Debian GNU/Linux 9 32-bit. The source code has been compiled with GCC 6.3.0 with `-O3` optimization enabled (see Table 6);

Tables 1, 2, 3, 4, 5, 6 show the time required to encrypt/decrypt one million of different plaintexts (fixed size of 128 bits) using the same key (randomly chosen). Notice that in addition to the key bits needed for the initial AddRoundKey AK^0, SPNbox layer γ uses 512 key bits — i.e. 512 bit = 16 round $\times$ 32 bit (n_{in}=32), or 512 bit = 32 round $\times$ 16 bit (n_{in}=16), or 512 bit = 64 round $\times$ 8 bit (n_{in}=8). Therefore, we set to 512 the minimum amount of key bits to be used in our solution. In particular, we will execute: 12 rounds ($R_{n_{in}}$=12), using 528 key bits (n_{in}=32); 24 rounds ($R_{n_{in}}$=24), using 528 key bits (n_{in}=16); and finally 47 rounds ($R_{n_{in}}$=47), using 517 key bits (n_{in}=8).

Table 1. results of tests on i3-330M with Ubuntu 18.04.1 LTS 64-bit.

	γ	γ **with BitShift**
n_{in}=32, encryption	1.178316 s	0.955048 s
n_{in}=32, decryption	1.447580 s	1.168507 s
n_{in}=16, encryption	3.946748 s	3.222751 s
n_{in}=16, decryption	4.193261 s	3.308678 s
n_{in}=8, encryption	2.547156 s	2.192452 s
n_{in}=8, decryption	2.564750 s	2.250102 s

Table 2. results of tests on i3-350M with Ubuntu 18.04.2 LTS 64-bit.

	γ	γ **with BitShift**
n_{in}=32, encryption	1.116117 s	0.902140 s
n_{in}=32, decryption	1.367435 s	1.150235 s
n_{in}=16, encryption	3.717744 s	3.035942 s
n_{in}=16, decryption	3.954781 s	3.116000 s
n_{in}=8, encryption	2.395998 s	2.061877 s
n_{in}=8, decryption	2.405397 s	2.114405 s

Table 3. results of tests on i7-2860QM with Kubuntu 18.10 64-bit.

	γ	γ **with BitShift**
n_{in}=32, encryption	0.837671 s	0.668838 s
n_{in}=32, decryption	0.925293 s	0.816856 s
n_{in}=16, encryption	2.667934 s	2.147471 s
n_{in}=16, decryption	2.811657 s	2.394600 s
n_{in}=8, encryption	1.886357 s	1.565764 s
n_{in}=8, decryption	2.030491 s	1.777118 s

Table 4. results of tests on i7-5500U with Ubuntu 18.04.2 LTS 64-bit.

	γ	γ **with BitShift**
n_{in}=32, encryption	0.861415 s	0.701899 s
n_{in}=32, decryption	0.954985 s	0.782088 s
n_{in}=16, encryption	2.980274 s	2.461575 s
n_{in}=16, decryption	3.155612 s	2.543056 s
n_{in}=8, encryption	1.860916 s	1.774127 s
n_{in}=8, decryption	1.879749 s	1.785562 s

Table 5. results of tests on i7-8550U with Ubuntu 18.04.2 LTS 64-bit.

	γ	γ **with BitShift**
n_{in}=32, encryption	0.681576 s	0.526522 s
n_{in}=32, decryption	0.723118 s	0.587942 s
n_{in}=16, encryption	2.396308 s	1.898987 s
n_{in}=16, decryption	2.462049 s	1.933232 s
n_{in}=8, encryption	1.160104 s	1.258072 s
n_{in}=8, decryption	1.179036 s	1.248327 s

Table 6. results of tests on i3-350M with Debian GNU/Linux 9 32-bit.

	γ	γ **with BitShift**
n_{in}=32, encryption	1.247818 s	1.041543 s
n_{in}=32, decryption	1.967226 s	1.558086 s
n_{in}=16, encryption	3.721377 s	3.381363 s
n_{in}=16, decryption	4.164744 s	3.262065 s
n_{in}=8, encryption	2.399780 s	2.065451 s
n_{in}=8, decryption	2.412146 s	2.127425 s

Our testing activities show that implementations with BitShift are generally faster than those without it. In particular, several cases show that the improvement in the execution time exceeds 20%. Only in Table 5, n_{in}=8, encryption and decryption, we find a different result.

Algorithm 2: MQTT: testing activity executed for each payload (16, 64, 128, and 1024 bytes)

```
Function layer_gamma_without_Bitshift(state, n_in, key):
    foreach chunk of length n_in in state do
        AddRoundKey(state, key)
        for i = 1 → R_{n_in} do
            SubBytes(state)
            MixColumns(state)
            AddRoundKey(state, key)
Function layer_gamma_with_Bitshift(state, n_in, key):
    foreach chunk of length n_in in state do
        AddRoundKey(state, key)
        for i = 1 → R_{n_in} do
            SubBytes(state)
            BitShift(state)
            MixColumns(state)
            AddRoundKey(state, key)
Function main():
    Open a connection
    Set key k = Random()
    Set plaintext p = Random()
    foreach n_in in (8, 16, 32) do
        Set timer t_1 = 0
        Set p_1 = p
        for i = 1 → 100 do
            for j = 1 → 10,000 do
                layer_gamma_without_BitShift(p_1, n_in, k)
            Send p_1 as payload with a MQTT message
        Stop timer t_1
        Set timer t_2 = 0
        Set p_2 = p
        for i = 1 → 100 do
            for j = 1 → 10,000 do
                layer_gamma_with_BitShift(p_2, n_in, k)
            Send p_2 as payload with a MQTT message
        Stop timer t_2
        Compare t_1 and t_2
        Close the connection
```

7.2. IoT environment

The testing activity has been performed using *MQTT* [65], a lightweight communication protocol designed for small sensors and mobile devices in low bandwidth environments. By default data are sent in clear text over the Internet, thus we encrypt data contained in the payload. We measure the performance of layer γ as described in Algorithm 2. More precisely, we compare the performances with and without BitShift transformation for different n_{in} — size of 32, 16, and 8 bits — encrypting one million of different plaintexts — size of 16, 64, 256, and 1024 bytes — using the same key. Then, we send one hundred MQTT messages, each of which contains 10,000 encrypted payloads. Finally, adopting the same approach, the server collects and decrypts the same number of MQTT messages with encrypted payloads.

Table 7. Encryption/decryption operations with a black-box implementation (server-side)

Payload	n_{in}	Encryption			Decryption		
(Bytes)	(Bits)	w/o BitShift	with BitShift	Gain	w/o BitShift	with BitShift	Gain
16	32	3.668s	3.319s	9.507%	0.893s	0.741s	16.999%
	16	6.335s	5.763s	9.037%	3.096s	2.412s	22.091%
	8	4.510s	4.882s	-8.254%	1.479s	1.551s	-4.929%
64	32	6.679s	5.601s	16.139%	3.636s	2.950s	18.865%
	16	14.869s	12.362s	16.860%	12.488s	10.023s	19.739%
	8	8.817s	9.616s	-9.060%	6.183s	6.446s	-4.253%
128	32	16.021s	13.847s	13.569%	14.166s	11.827s	16.512%
	16	51.098s	38.998s	23.680%	50.632s	39.596s	21.798%
	8	24.280s	25.709s	-5.884%	24.454s	25.825s	-5.607%
1024	32	54.047s	41.716s	22.816%	56.494s	47.065s	16.690%
	16	191.262s	151.101s	20.998%	195.424s	154.029s	21.182%
	8	92.651s	98.998s	-6.850%	93.744s	98.362s	-4.926%

Our testing activity has been executed on a machine equipped with an Intel® Core™ i7-6500U CPU @ 2.50GHz x 4 processor, with 12 GB SDRAM DDR4-2133, Intel® HD Graphics 520 (Skylake GT2) GPU and operating system Ubuntu 18.04.2 TLS. We used *Mosquitto* [66] version 1.4.15, which implements the MQTT protocol versions 3.1.1. The source code has been compiled with GCC 7.4.0, -O3 optimization enabled. Table 7 summarizes the results obtained.

In particular, for the encryption phase, we got a highest gain (23.680%) in the case of 128-byte payload and $n_{in} = 16$, while the highest loss (-9.060%) in the case of a 64-byte payload and $n_{in} = 8$. For the decrypt phase, the highest gain (22.091%) is obtained with a 16-byte payload and $n_{in} = 16$, and the highest loss (-5.607%) with a 128-byte payload and $n_{in} = 8$. Notice that the case $n_{in} = 8$ turned out to be the worst one.

8. Conclusions

In the era of Internet of Things the involved devices are usually lightweight, so they cannot perform heavy computations nor store a huge amount of data. In addition, these data might be sensitive — energy consumptions, medical records, and so on — and could be sent in an unprotected environment. In a white-box scenario, an attacker could easily read these data because she/he has full access to the whole execution platform and white-box cryptography can be used to secure data in this specific context.

Considering the effectiveness of side-channel attacks, new ciphers has been designed with white-box attack model in mind. In this paper, we focused on the SPNbox family [24], suggesting how to increase the number of key bits used in each round and showing that this improvement affects the performance of the cipher. The introduction of a key-dependent circular bit shift transformation helped us to increase the keyspace and to reduce the number of rounds of the cipher, reducing the execution time too.

We described and analyzed the performance of the modified cipher in the IoT context, where both white-box and black-box implementations may be required. In particular, we measured its performance (a) on 32/64-bit architectures and (b) encrypting the payload of an IoT messaging protocol. Our testing activities have been executed on consumer laptops. The results obtained encrypting and decrypting one million of different 128-bit plaintexts on 32/64-bit architectures showed that the execution time for layer γ is reduced up to 22% while the highest loss is about 8%.

Moreover, the testing activities performed with lightweight protocol MQTT got a gain of about 23% and 22% (encryption and decryption phase, respectively) while a loss of about 9% and 5%. In all our testing activities the case $n_{in} = 8$ turns out to be the worst one.

Possible future works are try to (a) understand in details why current implementation fails for $n_{in} = 8$ and (b) implement a communication protocol based on TLS-PSK, in order to compare the performance of white-box implementations with those of lightweight ciphers.

Author Contributions: All the authors contributed equally to the work. D.G.V.A., F.C., and A.S. wrote the code and executed testing activities; M.C., N.F., and A.V. wrote the manuscript; All the authors discussed the results and provided critical comments; A.V. supervised the testing activities.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ashton, K. That 'internet of things' thing. *RFID journal* **2009**, *22.7*, 97–114.
2. Harini, S.; Jothika, K.; Jayashree, K. A Survey on Privacy and Security in Internet of Things. *Int. Journ. of Innov. in Engin. and Tech.* **2017**, *8*, 129–134.
3. Bertino, E. Data Security and Privacy in the IoT. *EDBT* **2016**, *2016*, 1–3.
4. Tsai, C.; Lai, C.F.; Chiang, M.C.; Yang, L.T. Data mining for internet of things: A survey. *IEEE Comm. Surv. & Tut.* **2013**, *16.1*, 77–97 .
5. Schlesinger, M.; Parisi, D. The agent-based approach: A new direction for computational models of development. *Dev. Rev.* **2001**, *21.1*, 121–146.
6. Visconti, A.; Tahayori, H. Artificial immune system based on interval type-2 fuzzy set paradigm. *Appl. Soft Comput.* **2011**, *11.6*, 4055–4063.
7. Lee, S.; Kyon-Mo, Y.; Sung-Bae, C. Integrated modular Bayesian networks with selective inference for context-aware decision making. *Neurocomp.* **2015**, *163*, 38–46.
8. Visconti, A.; Tahayori, H. Detecting misbehaving nodes in MANET with an artificial immune system based on type-2 fuzzy sets. In *International Conference for Internet Technology and Secured Transactions*, Proceedings of the IEEE 4th International Conference for Internet Technology and Secured Transactions 2009, London, UK, 9 - 13 November 2009; IEEE.
9. Shi, Y.; Wei, W.; He, Z.; Fan, H. An ultra-lightweight white-box encryption scheme for securing resource-constrained IoT devices. In Proceedings of the 32nd Annual Conference on Computer Security Applications, Los Angeles, California, USA, 5 - 8 December 2016; ACM: New York, USA, 2016.
10. Shamir, A.; Van Someren, N. Playing "hide and seek" with stored keys. In Proceedings of the conference Financial Crypto 1999 (FC'99), Anguilla, BWI, 22 - 25 February 1999; Franklin M., Ed.; Springer: Heidelberg, Germany; pp. 118 –124.
11. Boneh, D.; DeMillo, R.A.; Lipton, R.J. On the importance of checking cryptographic protocols for faults (extended abstract). In Proceedings of the conference EUROCRYPT97, Konstanz, Germany, 11-15 May 1997; Fumy W., Ed.; Springer: Heidelberg, Germany; pp. 37–51.
12. Biham, E.; Shamir, A. Differential fault analysis of secret key cryptosystems. In Proceedings of the conference CRYPTO97, Santa Barbara, California, USA, 17 - 21 August 1997; Kaliski B.S., Ed.; Springer: Heidelberg, Germany; pp. 513–525.
13. Chow, S.; Eisen, P.; Johnson, H.; Van Oorschot, P.C. White-box cryptography and an AES implementation. In Proceedings of the International Workshop on Selected Areas in Cryptography 2002, John's, Newfoundland, Canada, 15 - 16 August 2002; Nyberg K., Heys H., Eds.; Springer: Berlin, Germany; pp. 250–270.
14. Cho, J.; Kyu Young, C.; Dukjae M. Hybrid WBC: Secure and efficient encryption schemes using the White-Box Cryptography. *IACR Cryptol. ePrint archive* **2015**, 2015, 800.
15. Chow, S.; Eisen, P.; Johnson, H.; Van Oorschot, P.C. A white-box DES implementation for DRM applications. In Proceedings of the ACM Workshop on Digital Rights Management, Washington, DC, USA, 18 November 2002; Feigenbaum J., Ed.; Springer: Berlin, Germany; pp. 1–15.
16. Billet, O.; Henri, G.; Charaf, E. Cryptanalysis of a white box AES implementation. In Proceedings of the International Workshop on Selected Areas in Cryptography, Waterloo, Canada, 9 - 10 August 2004; Handschuh H., Hasan M.A., Eds.; Springer: Berlin, Germany; pp. 227–240.

17. De Mulder, Y.; Roelse, P.; Preneel, B. Revisiting the BGE attack on a white-box AES implementation. *IACR Cryptol. ePrint archive* **2017**, 2013, 450.
18. De Mulder, Y.; Roelse, P.; Preneel, B. Cryptanalysis of the Xiao–Lai white-box AES implementation. In Proceedings of the International Conference on Selected Areas in Cryptography 2012, Windsor, ON, Canada, 15 - 16 August 2012; Knudsen L.R., Wu H., Eds.; Springer: Berlin, Germany; pp. 34–39.
19. Michiels, W.; Gorissen, P.; Hollmann, H.D.L. Cryptanalysis of a generic class of white-box implementations. In Proceedings of the International Workshop on Selected Areas in Cryptography 2008, Sackville, New Brunswick, Canada, 14 - 15 August 2008; Avanzi R.M, Keliher L., Sica F., Eds.; Springer: Berlin, Germany; pp. 414–428.
20. Lepoint, T.; Rivain, M.; De Mulder, Y.; Roelse, P.; Preneel, B. Two attacks on a white-box AES implementation. In Proceedings of the International Conference on Selected Areas in Cryptography 2013, Burnaby, BC, Canada, 14 - 16 August 2013; Lange T., Lauter K., Lisoněk P., Eds.; Springer: Berlin, Germany; pp. 265–285.
21. Jacob, M.; Boneh, D.; Felten, E. Attacking an obfuscated cipher by injecting faults. In Proceedings of the ACM Workshop on Digital Rights Management, 2002. Washington, DC, USA, 18 November 2002; Springer: Berlin, Germany; pp. 16–31.
22. Biryukov, A.; Bouillaguet, C.; Khovratovich, D. Cryptographic schemes based on the ASASA structure: Black-box, white-box, and public-key (Extended Abstract). In *Advances in Cryptology* , Proceedings of ASIACRYPT 2014, Kaohsiung, Taiwan, 7 - 11 December 2014; Sarkar, P., Iwata, T., Eds.; Springer: Berlin, Germany, 2014; pp. 63–84.
23. Bogdanov, A.; Takanori, I. White-box cryptography revisited: space-hard ciphers. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, Colorado, USA, 12 - 16 October 2015; ACM: New York, NY, USA, 2015; pp. 1050–1069.
24. Bogdanov, A.; Takanori I.; Tischhauser, E. Towards practical whitebox cryptography: optimizing efficiency and space hardness. In Proceedings of ASIACRYPT 2016, Hanoi, Viet Nam, 4 - 8 December 2016; Cheon J.H., Tsuyoshi T., Eds.; Springer: Berlin, Germany; pp. 126–158.
25. Shi, Y.; Wei, W.; Fan, H.; Au, M. H.; Luo, X. A Light-Weight White-Box Encryption Scheme for Securing Distributed Embedded Devices. *IEEE Trans. on Comp.* in press.
26. Biryukov, A.; Shamir, A. Structural cryptanalysis of SASAS. *Journ. of crypt.* **2014**, *23.4*, 505–518.
27. Cioschi, F.; Fornari, N.; Visconti, A. White-Box Cryptography: A Time-Security Trade-Off for the SPNbox Family. In Proceedings 2nd International Conference on Wireless Intelligent and Distributed Environment for Communication (WIDECOM 2019), Milan, Italy, 11 - 13 February 2019; Woungang I., Dhurandher S., Eds.; Springer: Cham; pp. 153–166.
28. Shannon, C.E. Communication theory of secrecy systems. *Bell syst. tech. journ.* **1949**, *28.4*, 656–715.
29. Feistel, H. Cryptography and computer privacy. *Sci. amer.* **1973**, *228.5*, 15–23.
30. Lee, S.; Jho, N. S.; Kim, M. A Key Leakage Preventive White-box Cryptographic Implementation. *IACR Cryptol. ePrint archive* **2018**, 2018, 1047.
31. Bringer, J.; Chabanne, H.; Dottax, E. Perturbing and protecting a traceable block cipher. In *Communications and Multimedia Security*, Proceedings of the 10th IFIP TC-6 TC-11 International Conference, Heraklion, Crete, Greece, 19-21 october 2006; Springer: Berlin, Heidelberg, Germany; pp. 109–119.
32. Bringer, J., Chabanne, H., Dottax, E. White box cryptography: Another attempt. *IACR Cryptol. ePrint archive* **2006**, 2006, 468.
33. De Mulder, Y.; Wyseur, B.; Preneel, B. Cryptanalysis of a Perturbated White-Box AES Implementation. In *Progress in Cryptology*, Proceedings of the conference INDOCRYPT 2010, Hyderabad, India, 12-15 December 2010; Gong G., Gupta K.C. Eds.; Springer: Berlin, Heidelberg, Germany; pp 292–310.
34. Karroumi, M. Protecting white-box AES with dual ciphers. In Proceedings of the International Conference on Information Security and Cryptology 2010, Seoul, Korea, 1 - 3 December 2010; Rhee K.H., Nyang D., Eds.; Springer: Berlin, Germany; pp. 278–291.
35. Xiao, Y.; Xuejia, L. A secure implementation of white-box AES. In Proceedings of the 2009 2nd International Conference on Computer Science and its Applications, Jeju, Korea (South), 10 - 12 December 2009; IEEE: 2009; pp. 1–6.
36. Luo, R., Lai, X., You, R. A new attempt of white-box AES implementation, Proceedings of the International Conference on Security, Pattern Analysis, and Cybernetics, Wuhan,China, 18-19 October 2014; IEEE; pp. 423–429.

37. Bai, K.; Wu, C.; Zhang, Z. Protect white-box AES to resist table composition attacks. *IET Information Security*, **2018**, *12(4)*, 305-3-13.
38. Derbez, P.; Fouque, P. A.; Lambin, B.; Minaud, B. On Recovering Affine Encodings in White-Box Implementations.In Proceedings of the Conference on Cryptographic Hardware and Embedded Systems 2016 (CHES2016), Santa Barbara, USA, August 17 –19 2016; Gierlichs B., Poschmann A.Y., Eds; Springer: Heidelberg, Germany; pp. 121–149.
39. Biryukov, A.; De Cannière, C.; Braeken, A.; Preneel, B. A toolbox for cryptanalysis: Linear and affine equivalence algorithms. In *Advances in Cryptology - EUROCRYPT 2003*, Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, 4-8 May 2003; Springer: Berlin, Germany, Heidelberg; pp.33–50.
40. Dinu, I. An improved affine equivalence algorithm for random permutations. In *Advances in Cryptology* , Proceedings of the conference EUROCRYPT 2018, Tel Aviv, Israel, April 29 - May 3 2018; Springer: Cham, Germany; pp. 413–442.
41. Xu, T.; Wu, C.; Liu, F.; Zhao, R. Protecting white-box cryptographic implementations with obfuscated round boundaries. Science China Information Sciences, **2017**, *61(3)*.
42. Yongjin, Y.; Dong-Chan, K.;Hun, B. C.,; Junbum, S. Cryptanalysis of the Obfuscated Round Boundary Technique for Whitebox Cryptography. *Science China Information Sciences*.
43. Rivain, M.; Wang, J. Analysis and Improvement of Differential Computation Attacks against Internally-Encoded White-Box Implementations. *IACR Trans. on Crypt. Hardware and Embedded Systems.* **2019**, 225–255.
44. Bos, J. W.; Hubain, C.; Michiels, W.; Teuwen, P. Differential computation analysis: Hiding your white-box designs is not enough. In Proceedings of the Conference on Cryptographic Hardware and Embedded Systems 2016 (CHES2016), Santa Barbara, USA, August 17 –19 2016; Gierlichs B., Poschmann A.Y., Eds; Springer: Heidelberg, Germany; pp. 215–236.
45. Dusart, P.; Letourneux, G.; Vivolo, O. Differential fault analysis on AES. In Proceedings of the International Conference on Applied Cryptography and Network Security 2003, Kunming, China, 16 - 19 October 2003; Zhou J., Moti Y., Han Y., Eds.; Springer: Berlin, Germany; pp. 293–306.
46. Kocher, P.; Jaffe, J.; Jun, B.; Rohatgi, P. Introduction to differential power analysis. *Journ. of Crypt. Engin.* **2011**, *1.1.*, 5–27.
47. Alpirez Bock, E.; Bos, J.W.; Brzuska, C.; Hubain, C.; Michiels, W.; Mune, C.; Sanfelix Gonzalez, E.; Teuwen, P.; Treff, A. White-Box Cryptography: Don't Forget About Grey Box Attacks. *IACR Cryptol. ePrint archive* **2017**, 2017, 355.
48. Bock, E.A.; Brzuska, C.; Michiels, W.; Treff, A. On the ineffectiveness of internal encodings - revisiting the DCA attack on white-box cryptography. In Proceedings of the 16th International Conference on Applied Cryptography and Network Security (ACNS2018), Leuven, Belgium, 2-4 July 2018; Preenel B., Vercauteren F., Eds.; Springer: Heidelberg, Germany; pp. 103–120.
49. Biryukov, A.; Udovenko, A. Attacks and countermeasures for white-box designs. In International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, 8-12 December 2019; Springer: Cham, Germany; pp. 373–402.
50. Lee, S.; Kim, T.; Kang, Y. A masked white-box cryptographic implementation for protecting against differential computation analysis. *IEEE Transactions on Information Forensics and Security*, **2018**, *13(10)*, 2602–2615.
51. Banik, S.; Bogdanov, A.; Isobe, T.; Jepsen, M. Analysis of software countermeasures for whitebox encryption. *IACR Transactions on Symmetric Cryptology*, **2017**, 307–328.
52. Marin, L. White Box Implementations Using Non-Commutative Cryptography. *Sensors*, **2019**, *19.5*.
53. Bogdanov, A.; Rivain, M.; Vejre, P.S.; Wang, J. Higher-order DCA against standard side-channel countermeasures. *IACR Cryptol. ePrint archive* **2018**, 2018, 869.
54. Fouque, P.A.; Karpman,P.; Kirchner, P.; Minaud, B. Efficient and provable white-box primitives. In *Advances in Cryptology* , Proceedings of ASIACRYPT 2016, Hanoi, Viet Nam, 4 - 8 December 2016; Cheon J. H., Takagi T. Eds.; Springer: Berlin, Heidelberg, 2016; pp 159 – 188
55. Biryukov, A.; Perrin, L. Symmetrically and Asymmetrically Hard Cryptography. Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Hong Kong, China, 3 - 7 December 2017; Springer: Cham, Germany; pp. 417–445

56. Bock, E. A.; Amadori, A.; Bos, J. W.; Brzuska, C.; Michiels, W. Doubly half-injective PRGs for incompressible white-box cryptography. In Cryptographers' Track at the RSA Conference, S. Francisco, California, USA, 4 - 8 March 2019; Springer, Cham, Germany; pp. 189-209.
57. Feng, Q.; He, D.; Wang, H.; Kumar, N.; Choo, K. K. R. White-Box Implementation of Shamir's Identity-Based Signature Scheme. *IEEE Systems Journal*, **2019**.
58. Goubin, L.; Paillier, P.; Rivain, M.; Wang, J. How to reveal the secrets of an obscure white-box implementation. *Journal of Cryptographic Engineering*, **2018**, 1-18.
59. Moriarty, K.; Kaliski, B.; Rusch, A. PKCS# 5: Password-Based Cryptography Specification Version 2.1. *Internet Requests for Comments.* **2017**, RFC 8018. Available online: https://tools.ietf.org/html/rfc8018 (accessed on 3 June 2019).
60. Visconti, A.; Bossi, S.; Ragab, H.; Calò, A. On the weaknesses of PBKDF2. In *Cryptology and Network Security,* Proceedings of the 14th International Conference, CANS 2015, Marrakesh, Morocco, 10 - 12 December 2015; Reiter M., Naccache D., Eds.; Springer International Publishing: Switzerland , 2015; pp. 119–126.
61. Visconti, A.; Mosnáček, O.; Brož, M.; Matyáš, V. Examining PBKDF2 security margin—Case study of LUKS. *Journ. of Inf. Sec. and Appl.* **2019**, *46*, 296–306.
62. Visconti, A.; Gorla, F. Exploiting an HMAC-SHA-1 optimization to speed up PBKDF2. *IEEE Transactions on Dependable and Secure Computing.* **2018**.
63. Biryukov, A.; Dinu, D.; Khovratovich, D. Argon2 (version 1.2) Cited 13 Nov 2018. Available online: https://password-hashing.net/submissions/specs/Argon-v3.pdf (accessed on 28 May 2019).
64. Percival, C.; Josefsson, S. The scrypt Password-Based Key Derivation Function. *Internet Requests for Comments*. **2016**, RFC 7914, 16. Available online https://tools.ietf.org/html/rfc7914 (accessed on 30 May 2019).
65. Banks, A.; Gupta, R. MQTT Version 3.1.1 Plus Errata 01. Available online: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html (accessed on 24 May 2019).
66. Light, R. A. Mosquitto: server and client implementation of the MQTT protocol. *J. Open Source Software*. **2017**, *2*, 265–265.