Evaluation of white-box and grey-box Noekeon implementations in FPGA

Zouha Cherif\textsuperscript{1,2}, Florent Flament\textsuperscript{1}, Jean-Luc Danger\textsuperscript{1}, Shivam Bhasin\textsuperscript{1}, Sylvain Guilley\textsuperscript{1} and Hervé Chabanne\textsuperscript{1}

\textsuperscript{1}Institut TELECOM, TELECOM ParisTech, CNRS LTCI, 46 rue Barrault 75634 Paris, France.
\textsuperscript{2}Institut TELECOM, École Supérieure des Communications de Tunis Cité Technologique des Communications 2083 Ariana, Tunisie.
\texttt{<firstname.lastname@telecom-paristech.fr>}

\textbf{Abstract}—White-box implementations of cryptographic algorithms aim to denying the key readout even if the source code embedding the key is disclosed. They are based on sets of large tables perfectly known by the user but including unknown encoding functions. While former white-box implementations have been proposed in software, hardware white-box implementations are also possible. Their main drawback is the complexity of their architectures, which often requires large tables. In this paper we show that it is possible to implement white-box cryptography in an FPGA by taking advantages of LUTs. We also propose a grey-box approach, where intermediate random variables are unknown to the attacker. We show that such approach allows to reduce the complexity by using fewer tables. The resistance against side channel attacks has been evaluated for different implementations. Our results show the interest of the proposed methods for a better compromise complexity/security.

\textbf{Index Terms}—white-box cryptography, grey-box cryptography, FPGA implementations, Noekeon, Side Channel Analysis, SCA, Mutual Information Metric, MIM, random number generator, TRNG.

\section{I. Introduction}

Side channel analysis or attacks (SCA) exploit information leaked from the physical implementation of a cryptographic system. The leakage is passively observed via timing information, power consumption, electromagnetic radiations, \textit{etc}. Protection against side channel attacks is important because the attacks can be implemented quickly and at a low cost.

White-box cryptography has been introduced in the domain of Digital Rights Management with the ambitious goal of protecting keys of a block cipher while leaving to an adversary the whole access to the software implementing this algorithm. Practically, this leads to ciphers represented by a network of look-up tables (LUTs), which hide the structure of the cipher as the tables embed external encoding bijections. Software white-box implementations for DES and AES have been proposed in [1], [2], while several cryptanalytic works have been done on such implementations [3]–[6]. Then, a hardware grey-box implementation has been introduced in [7] to thwart Side Channel Attacks against Reverse Engineering (SCARE) [8]. The grey-box approach is a degradation of the white-box approach, where some variables are random, hence reputed unknown. As in the case of the software white-box implementations, the whole computation is done inside tables, whose output are systematically encoded. In this paper, we consider that an attacker has access only to the bitstream and the power consumption of the device. We note that the bitstream does not reveal the key drown into the table, which may leak during computation. However, the grey-box implementation uses random bits that cannot be known by the attacker and that dynamically change the encoding functions used to encode the data. In this work we focus on key recovery. We will therefore experimentally evaluate the greybox hardware implementation of the Noekeon cipher that has been proposed by J. Bringer et al. in [7] and we will propose a white-box alternative using 3-bits encoding functions.

As the complexity of architectures based on large tables can be a major issue, we give some feasibility analysis of both white-box and grey-box implementations. To illustrate our results, we implemented the Noekeon cipher [9] as a reference. Nevertheless, without loss of generality, the proposed architectures may be ported to classical symmetrical cryptography algorithms as DES or AES. Thereafter we also evaluate the robustness of our implementations on FPGA with regards to security. We use mutual information metric as a distinguisher to characterize the point and amount of leakage without mounting an attack.

The remainder of this paper is organized as follows. Section II gives an overview of the Noekeon cipher. Section III focuses on practical aspects of a hardware white-box implementation. Section IV considers the grey-box implementation where random variables are used. Then section V presents a complexity comparison between the different implementations and assesses their robustness. Finally the conclusion, in section VI, summarizes the impact of the white-box and grey-box approaches to protect hardware implementations of cryptoprocessors.

\section{II. Noekeon in Hardware}

\textbf{A. The Noekeon Algorithm}

This section gives a short overview of the Noekeon cipher. Noekeon was proposed to the NESSIE project in 2000 [9]–[11]. Noekeon is a 128-bit block cipher over 16 rounds. It...
maintains a state of four 32-bit words: \( a_0, a_1, a_2, a_3 \). Each round is constituted by the following operations:

1) A first round constant is XORed to \( a_0 \).
2) A linear transformation \( \theta \) is applied to the four words \( a_0, a_1, a_2, a_3 \). During the execution of \( \theta \), the round key is XORed to the internal data of \( \theta \).
3) A second round constant is XORed to \( a_0 \).
4) \( \pi_1 \): The words \( a_1, a_2 \) and \( a_3 \) are circularly rotated of 1, 5, and 2 bits, respectively, to the left.
5) \( \Gamma \): All bits in the same position in \( a_0, a_1, a_2 \) and \( a_3 \) are grouped together in nibbles (i.e. words of 4 bits) which go through the same non-linear bijection \( \gamma \) (i.e. \( \gamma \) is applied 32 times, once for each nibble).
6) \( \pi_2 \): The words \( a_1, a_2 \) and \( a_3 \) are circularly rotated of 1, 5 and 2 bits, respectively, to the right.

Finally, after the last round, a final constant is XORed to \( a_0 \) and \( \theta \) is applied.

We first consider a basic hardware implementation of the Noekeon algorithm, where it has been naively described in a hardware description language. After synthesis, the complexity of our implementation’s datapath is 314 LUTs, including a linear stage of 207 LUTs and an non-linear stage of 164 LUTs.

III. NOEKEON HARDWARE WHITE-BOX IMPLEMENTATIONS

A. Hardware white-box implementation and its constraints

Chow et al. proposed software white-box implementations for the DES and AES cryptographic algorithms [1], [2]. The principles of these implementations can also be applied in hardware. The idea is to drown a cryptographic function with its key and some bijective encoding functions in a single lookup table. It can be applied in hardware implementations and especially in FPGAs, which are composed of many small LUTs (4 → 1 or 6 → 1 on recent FPGAs [12], [13]).

The word size on which the bijections are applied is very important if we want to be able to use general encoding functions. Indeed, using bit per bit bijections highly facilitates the design but causes issues that are addressed in the section IV-A. When using 2-bits words encoding functions, only one bit of out two can be protected. The best 2-bits words bijections are of the following form:

- \( a \rightarrow a \)
- \( b \rightarrow a \oplus b \)

On the other hand, considering too big words will not be efficient in term of size of LUTs. For instance, let's consider a simple function 2 bits to 1 bit, using 5-bits encoded words. If the 2 bits on which we want to compute our function are in two different encoded words and if we want that words be always encoded outsides lookup tables, then our only solution is to use a 10 → 5 LUT. It will decode the two 5-bits input words, then extract the two bits of interest, compute the result bit and finally put it in a 5-bits output word. This means that our 2 → 1 function’s size will be \( 5 \times 2^{10} = 5120 \) bits! Moreover, as described in [14], the usage of large LUTs facilitates DPA attacks.

A good compromise, in order to use generic encoding functions in hardware implementations, is to use 3-bits words. Several generic bijections with good protection properties can be implemented on 3-bits words. In that case, any 2 bits to 1 bit function can be built using one \( 6 \rightarrow 3 \) LUT, which is very interesting since most modern FPGAs are mainly made of \( 6 \rightarrow 1 \) lookup tables. To compare the size of such LUTs to the previous ones, a \( 6 \rightarrow 3 \) LUT has a size of \( 3 \times 2^6 = 192 \) bits.

Any linear function can be decomposed in 2 → 1 functions. However, the Noekeon algorithm contains 4 → 4 substitution boxes that cannot be implemented with 2 → 1 functions. In worst case, each of the four input bit is encoded in a distinct 3-bit word. This means that the LUTs implementing the sboxes should have 4 3-bits words (= 12 bits) as input. In that case, computing the 4 output bits (\( d_{01}, d_{11}, d_{12}, d_{22} \)) of a 4 → 4 substitution box would require two 12 → 3 LUTs. Since we want to avoid the use of LUTs with more than 6 inputs, we will see in the next section how we can deal with that issue by smartly reorganizing our 3-bits words.

B. Noekeon 3-bits encoding white-box implementation

The Noekeon algorithm can be efficiently implemented in FPGA with generic bijective encoding functions on 3-bits words. We will focus our study on the Noekeon round function, made of a first linear diffusion stage, followed by a non linear confusion stage. However, the robustness of this implementation has not been practically assessed in this work.

The linear stage can be decomposed in 2 → 1 XOR functions. Each LA (Linear Application) output bit can be computed independently from 7 input bits. However, it is possible to optimize the number of LUTs by smartly grouping the bits. If \( a_0, a_1, a_2, a_3 \) are the 4 32-bits words in the Noekeon state register as described in the Noekeon specification [9], and \( a_{i,j} \) is the \( j^{th} \) bit of the \( a_i \) 32-bits word, then for each \( j \in \{0, ..., 31\}, \) we can group \( a_{0,j} \) with \( a_{2,j} \) and \( a_{1,j} \) with \( a_{3,j} \) in two 3-bits word. Therefore, a pre-LA bits grouping manipulation that requires 32 <i>\text{l}\text{c}3</i> LUTs has to be implemented.

As said previously, each LA output bit can be computed independently. Let’s call \( b_{i,j} \) the outputs bits from LA where \( i \) can take its value in \{0, ..., 3\} and \( j \in \{0, ..., 31\}. \) Then:

\[
\begin{align*}
\text{\( b_{0,j} \)} &= a_{0,j} \oplus a_{1,j} \oplus a_{3,j} \oplus a_{1,j+8} \oplus a_{3,j+8} \oplus a_{1,j+24} \oplus a_{3,j+24} \oplus K_{0,j}; \\
\text{\( b_{1,j} \)} &= a_{0,j} \oplus a_{1,j} \oplus a_{2,j} \oplus a_{0,j+8} \oplus a_{2,j+8} \oplus a_{0,j+24} \oplus a_{2,j+24} \oplus K_{0,j+1}; \\
\text{\( b_{2,j} \)} &= a_{2,j} \oplus a_{1,j} \oplus a_{3,j} \oplus a_{1,j+8} \oplus a_{3,j+8} \oplus a_{1,j+24} \oplus a_{3,j+24} \oplus K_{1,j+1}; \\
\text{\( b_{3,j} \)} &= a_{0,j} \oplus a_{2,j} \oplus a_{3,j} \oplus a_{0,j+8} \oplus a_{2,j+8} \oplus a_{0,j+24} \oplus a_{2,j+24} \oplus K_{3,j};
\end{align*}
\]

\( K_{i,j} \) are constants that are computed once during the LUTs generations process according to the encryption key. Each bit \( b_{i,j} \) can be computed using the structure described on figure 1 (inputs bits must be taken as described by the previous AL equation). The unused output bits are computed using a
random function of the input bits. Therefore we need 128 such structures in order to compute every $b_{ij}$ bit.

The non-linear stage is composed of thirty two $4 \rightarrow 4$ substitution boxes, where each of them can be implemented with two $6 \rightarrow 3$ LUTs by smartly reorganizing our data with a pre-NA (Non-linear Application) bits manipulation. Now we consider the 4-bits nibbles that have to be processed by our substitution tables, let's call them $c_0$, $c_1$, ..., $c_{31}$. And $c_{ij}$ is the $j^{th}$ bit of the $i^{th}$ 4-bits nibble. The correspondence between $c_{ij}$ and $b_{ij}$ is defined by the $\pi_1$ function [9]. Then our pre-NA manipulation consists in grouping $c_{00}$ with $c_{10}$ and $c_{20}$ with $c_{30}$ in 2 3-bits encoded words. This manipulation has to be done for each of the 32 4-bits nibbles. Once the grouping has been done, each sbox can be implemented using two $6 \rightarrow 3$ LUTs. In order to compute every $d_i$, we need 32 such structures.

The NA output $d_i$ nibbles can be stored in the state register after being applied the pre-LA transformation. Additionally to reorganising the 3-bits words, the pre-LA transformation does the $\pi_2$ wires crossing [9] without additional cost. To complete our Noekeon datapath, an initial and a final transformation have to be added respectively at the input and the output of the datapath. These transformations are similar to the pre-LA and pre-NA transformations except that the initial transformation does not need decoding bijections at its input and the final bijection does not need encoding bijections at its output. The full Noekeon white-box implementation datapath is illustrated by figure 2, where IT and FT are respectively the initial and final transformations and $f_0$, $f_1$ and $f_2$ symbolize the 3-bits based encoding bijections.

We can now summarize the overall complexity of this implementation in term of number of LUTs according to the different stages described in this section.

- The pre-LA stage is composed of $2^4 \times 32 = 64 \rightarrow 3$ LUTs
- The LA stage is composed of $3 \times 128 = 384 \rightarrow 3$ LUTs
- The pre-NA stage is composed of $2^4 \times 32 = 64 \rightarrow 3$ LUTs
- The NA stage is composed of $2^4 \times 32 = 64 \rightarrow 3$ LUTs
- The IT stage is composed of $2^4 \times 32 = 64 \rightarrow 3$ LUTs
- The FT stage is composed of $2^4 \times 32 = 64 \rightarrow 3$ LUTs

When we sum up the LUTs of all these stages we obtain 704 $6 \rightarrow 3$ LUTs, or 2112 $6 \rightarrow 1$ LUTs.

IV. Grey-box and Noekeon

A. White-box 1-bit encoding drawbacks

As per the constraints induced by the use of general encoding bijections, as seen in section III, we explore in this section the advantages and disadvantages of using 1-bit encoding bijections. The only possible 1-bit (or bitwise) boolean bijection is the XOR operation with a constant word. The main advantage of such function is that encoded data do not have to be decoded by block as it is the case with blocks encoded using generic bijections. 1-bit encoded blocks can be split without any constraints. Such property is very useful when implementing cryptographic algorithms, since wires crossing are common operations among those algorithms. However, 1-bit encoding functions have their drawbacks.

To understand the 1-bit encoding drawback, let’s consider an unprotected Noekeon implementation. Let $S_0$ and $S_1$ be the unprotected data in the state register before and after the first encryption round. When the state register samples the $S_1$ value, the circuit’s power consumption is correlated to the hamming distance between $S_0$ and $S_1$, which can be predicted by an SCA attacker. Now we consider the static 1-bit encoding implementation, where data is always xored by a constant word $m$ before being stored in the state register. The circuit’s power consumption is correlated to the hamming distance between $S_0 \oplus m$ and $S_1 \oplus m$. When calculating the value of $HD(S_0 \oplus m, S_1 \oplus m)$, we find out that it is equals to $HD(S_0, S_1)$, the power consumption of the unprotected implementation.

$$HD(S_0 \oplus m, S_1 \oplus m) = HW(S_0 \oplus m \oplus S_1 \oplus m)$$
$$= HW(S_0 \oplus S_1)$$
$$= HD(S_0, S_1)$$

Therefore, the white-box 1-bit encoding implementation does not protect our cryptographic algorithm against SCAs based on the hamming distance model. Grey-box implementations have been proposed to cope with that issue.

B. Grey-box principles and FPGA implementation

A first hardware grey-box implementation was proposed for the Noekeon cryptographic algorithm in [7]. Whereas in the case of white-box implementations the encoding functions
Let’s try to theoretically assess the robustness of the grey-box implementation, by considering the S register’s power consumption in the four possible cases. $S_0$ and $S_1$ are the values stored in the state register before and after the first encryption round. $f_0$ and $f_1$ are two masks used to encode our data.

- $HD(S_0 + f_0, S_1 + f_1) = HW(S_0 + S_1 + f_0 + f_1)$
- $HD(S_0 + f_1, S_1 + f_0) = HW(S_0 + S_1 + f_0 + f_1)$
- $HD(S_0 + f_0, S_1 + f_0) = HW(S_0 + S_1)$
- $HD(S_0 + f_1, S_1 + f_1) = HW(S_0 + S_1)$

Now consider that an attacker is able to predict the value of $HW(S_0 + S_1)$. We can observe that when encoding masks are randomly chosen between $f_0$ and $f_1$, the attacker has a probability $\frac{1}{2}$ to correctly guess the power consumption value, while he has a probability $\frac{1}{2}$ to have more or less correlation between his prediction and the power consumption, when it equals $HW(S_0 + S_1 + K)$, $K$ being a constant. However, if we choose our masks $f_0$ and $f_1$ so that $f_1 = f_0$, then the attacker will not anymore be able to obtain a correlation between his predictions and the circuit’s power consumption. In that case, the attacker will be able to predict $HW(S_0 + S_1)$ while the power consumption will be either $HW(S_0 + S_1)$ or $HW(S_0 + S_1)$ with probability $\frac{1}{2}$.

Moreover, when dealing with random masks based protections, one has to consider the amount of entropy that is involved. Indeed, many countermeasures have already been proposed, that base or enhance their security on the usage of a random bit, like in random switching logic (RSL [15],[16]). Nonetheless, it has been observed that if the average value of the leakage carries no sensitive information, the distribution of the leakage could enable the prediction of the mask value [17]. The folding attack on MDPL is studied in practice in [18] and can be carried out effectively because the distribution allows to clearly identify the value of the mask bit. It is therefore paramount, when designing a countermeasure that involves shared randomness, to use more than one single bit of entropy.

Actually, one step further is accomplished in this article [19], leading to the conclusion that a fair number of random bits must be injected in the datapath for the masking not to be defeated in a straightforward manner.

C. Noekeon hardware grey-box implementation

The grey-box Noekeon implementation presented in [7] includes a mixing bijection before the linear application, as introduced by Chow et al. [2]. However, since the dynamic encoding functions are efficient to protect the Noekeon implementation against SCA, we could remove the mixing bijection from our design without reducing the circuit’s robustness. Therefore, the linear application stage of the Noekeon datapath can be implemented using 32 blocks of three $4 \rightarrow 4$ LUTs and four $3 \rightarrow 1$ LUTs. The non-linear application stage is composed of thirty two $4 \rightarrow 4$ substitution boxes, which can be implemented in thirty two $4 \rightarrow 4$ LUTs. The $\pi_1$ and $\pi_2$ bit crossing operations are costless since they are implemented by appropriate wires routing.

Let’s introduce the dynamic encoding bijections. In order to add 2 random bits to each LUT, as specified in the section IV-B, we can replace each $4 \rightarrow 4$ LUT by a $6 \rightarrow 4$ LUT. This has no cost on recent FPGAs, since they are made of $6 \rightarrow 1$ LUTs. In the same way, $3 \rightarrow 1$ functions can be implemented in $6 \rightarrow 1$ LUTs. The full Noekeon grey-box implementation datapath is illustrated by figure 4. $f_0$, $f_1$, $g_0$, $g_1$, $h_0$ and $h_1$ representing 1-bit encoding bijections.

Moreover, like in the case of white-box implementations, we have to use an initial and a final transformation. We can now summarize the overall complexity in terms of number of LUTs of our grey-box implementation:

- The $1^{st}$ layer of the LA stage is composed of $3*32=96$ $6 \rightarrow 4$ LUTs.
• The 2nd layer of the LA stage is composed of 4*32=128 LUTs.
• The NA stage is composed of 32 6 → 4 LUTs.
• The IT stage is composed of 32 6 → 4 LUTs.
• The FT stage is composed of 32 6 → 4 LUTs.

We have to add the complexity of the RGN component which is 100 LUTs 4 → 1. When we sum up the number of LUTs used in all these stages we obtain 192 6 → 4 LUTs, 128 6 → 1 LUTs and 100 6 → 1 LUTs, for a total of 996 6 → 1 LUTs. For the full implementation, we use 3 random bits. Each one is used twice, for example, we use the same random bit for the output of the 1st layer LA and the input of the 2nd layer LA. It is possible to use up to as much random bits as the number of 6 → 1 LUTs: 996 bits.

V. NOEKEON IMPLEMENTATIONS ASSESSMENT
A. Spatial complexity comparison

In this section we compare the spatial complexity, in term of number of lookup tables, of four implementations of the Noekeon algorithm. We consider the following implementations:

• A naive hardware implementation without any countermeasure that we call Basic hardware implementation.
• A hardware white-box implementation using 1-bit static encoding functions that we call White-box 1-bit implementation.
• A hardware white-box implementation using 3-bits static encoding functions that we call White-box 3-bits implementation.
• A hardware grey-box implementation using 1-bit dynamic encoding functions that we call Grey-box implementation.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>LA complexity</th>
<th>NA complexity</th>
<th>Total Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic hardware impl.</td>
<td>207</td>
<td>164</td>
<td>414</td>
</tr>
<tr>
<td>White-box 1-bit impl.</td>
<td>512</td>
<td>128</td>
<td>896</td>
</tr>
<tr>
<td>Grey-box impl.</td>
<td>512</td>
<td>128</td>
<td>996</td>
</tr>
<tr>
<td>White-box 3-bit impl.</td>
<td>1344</td>
<td>384</td>
<td>2112</td>
</tr>
</tbody>
</table>

Table I details the complexity of each implementation’s LA stage, NA stage and overall datapath. Unsurprisingly, the basic implementation is the smallest; the white-box 1-bit implementation and the grey-box one have the same complexity, but the white-box 3-bits implementation is more than twice the size of 1-bit implementations. While the white-box 1-bit implementation uses 4 → 1 LUTs, the grey-box one requires 6 → 1 LUTs; but since modern FPGAs are made of 6 → 1 LUTs, both implementation uses the same amount of LUTs. On the other hand, the white-box 3-bits implementation has a more complex architecture and therefore requires additional LUTs. Note that this spatial complexity estimation is biased, since the grey-box implementation requires a random number generator. Ideally, a true RNG (TRNG) should be used as the one described in [20]. Therefore in order to fairly compare the complexity of these implementations, one has to include the size of the number generator used. However, this has not been taken in account in this work.

B. Side-channel robustness assessment

Apart from evaluating the spatial complexity of the Noekeon hardware cryptoprocessor it is important to analyze the robustness w.r.t security. We analyze the robustness of Noekeon hardware implementations by calculating the mutual information which we try to use as an indicator of the robustness of the design. We would like to specify that mutual information is used as a tool to measure the amount of data-dependant leakage and we do not implement an attack. Therefore all the analysis is done knowing the value of secret key. In other words, mutual information graphs depict the level of dependency between the acquired traces and known internal values. Higher the dependency, lower the robustness of the target.

The three Noekeon implementations analyzed are the basic, the white-box 1-bit and the grey-box implementations. We synthesized the design of SASEBO-B boards which has an Altera Stratix II (EP2S30) using an 54855 Infinium oscilloscope from Agilent Technologies. Each trace was averaged 64 times. We collect 10000 traces for each of the three implementations. The white-box 3-bit implementation was not part of this study as we did not practically implement it on the FPGA as mentioned before. This is a part of further work in this context.

In order to compute the mutual information on our implementations, we defined a power consumption model. We use the hamming distance of the transition word in the state register at the moment of the first encryption round sampling. We note this $HD = HW(S_0 + S_1)$. The Noekeon datapath is 128 bits which means $0 \leq HD \leq 128$. We partition the traces in two groups: one with $HD \leq 64$ and the other with $HD > 64$. More than two partition will require more traces for reliable results. We compute the mutual information trace $MI(O; HD)$ between our observations and the known hamming distances $HD$, computed according to the secret key and the messages encrypted.

Figure 5, shows the mutual information curve for the unprotected implementations using a correct key and three random false keys. It is evident from the figure 5 that the correct key result in higher mutual information than the false key. This also means that if an implementation is not robust and leaks sensitive information, mutual information corresponding to correct key should be highest. This is a special case of mutual information analysis [21], where we observe the mutual information for the correct key only. Figure 6 displays the mutual information traces of each implementation calculated using the correct key for 10,000 traces. When a peak appears, it means that the corresponding implementation leaks information on the secret key. The unprotected and the white-box 1-bit implementations leak almost the same amount of information, whereas the grey-box implementation does...
not leak any information according to the power consumption model considered.

VI. CONCLUSION

In this article, we assess hardware white-box and grey-box implementations’ complexity and robustness against SCA. We first propose a hardware white-box implementation of the Noekeon encryption algorithm using 3-bit generic encoding functions and compare its complexity to three other implementations. While a basic unprotected hardware implementation of the Noekeon datapath requires 414 LUTs, our white-box implementation using 1-bit encoding functions uses 896 LUTs, our grey-box implementation use 996 LUTs; the white-box implementation using 3-bit generic encoding functions uses 2112 LUTs.

Moreover, we evaluate the practical robustness of our designs using the mutual information. White-box 1-bit implementation not as robust as the unprotected one despite using more the resources because of a flaw in its architecture that we described. On the other hand, the grey-box implementation, which uses dynamic encoding functions, does not leak information according to our mutual information results and is therefore resistant against SCA. Since these results should be valid for any cryptographic algorithm like AES and DES, the low complexity combined with the high robustness against SCA of the grey-box implementation detailed in this article make such implementation a good candidate for SCA resistant cryptographic algorithms implementations. However, grey-box implementations rely on random number generators, ideally TRNGs, whose complexity must also be taken in account. The proposed white-box 3-bits implementation using static generic encoding functions will be assessed in a further work.

REFERENCES