White-Box Cryptography in the Gray Box
– A Hardware Implementation and its Side Channels –

Pascal Sasdrich[1(✉)], Amir Moradi[1], and Tim Güneysu[2]

[1] Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Bochum, Germany
{Pascal.Sasdrich,Amir.Moradi}@rub.de
[2] University of Bremen and DFKI, Bremen, Germany
tim.gueneysu@uni-bremen.de

Abstract. Implementations of white-box cryptography aim to protect a secret key in a white-box environment in which an adversary has full control over the execution process and the entire environment. Its fundamental principle is the map of the cryptographic architecture, including the secret key, to a number of encoded tables that shall resist the inspection and decomposition of an attacker. In a gray-box scenario, however, the property of hiding required implementation details from the attacker could be used as a promising mitigation strategy against side-channel attacks (SCA). In this work, we present a first white-box implementation of AES on reconfigurable hardware for which we evaluate this approach assuming a gray-box attacker. We show that – unfortunately – such an implementation does not provide sufficient protection against an SCA attacker. We continue our evaluations by a thorough analysis of the source of the observed leakage, and present additional results which can be used to build stronger white-box designs.

1 Introduction

Initially the field of white-box cryptography was mainly motivated by applications of the field of Digital Rights Management (DRM) that aims to protect a secret key in a white-box environment, where an adversary has full control over the execution process and the environment of a cryptographic implementation. However, with the widespread emerging of embedded and pervasive computing devices implementing cryptographic functions and primitives, the threat of white-box adversaries is no longer limited to cryptographic software implementations. Although, an adversary might be limited by the gray-box model in practice (i.e., he cannot control the execution process and the environment entirely), Side-Channel Analysis (SCA) attacks are well-known to be used to exploit information leakage related to the device internals e.g., by analyzing power consumption or electromagnetic radiations (EM). Still, for successfully mounting such physical attacks, the attacker requires at least some knowledge about the internals in order to build adequate hypotheses that can be used, for example, for key extraction. In this context the nature of white-box cryptography that effectively disguising all internals and the secret key from the attacker

T. Peyrin (Ed.): FSE 2016, LNCS 9783, pp. 185–203, 2016.
DOI: 10.1007/978-3-662-52993-5_10

by encoding them into tables, seems to yield some inherent resistance against such physical attacks.

Previous Works: In 2002, first white-box implementations for DES [9] and AES [10] were proposed by Chow et al. in order to protect a secret key within a cryptographic implementation in presence of a white-box adversary. However, these seminal proposals and their implementations were soon shown to be vulnerable to differential cryptanalysis [13,22] as well as algebraic cryptanalytic attacks [3,16,17]. This led to some new proposals for white-box implementations of AES. In 2009, Xiao et al. in [23] proposed a variant of the design of Chow et al. using larger linear encodings, for which again a vulnerability against algebraic cryptanalytic attacks was identified in [20]. Other approaches suggest to build white-box AES implementations using pertubations [7] (which was broken in [21]) or based on the concept of dual-ciphers [14].

Recent work in [2] aims to generalize and formalize notions for white-box cryptography and related attacks for any SLT cipher presenting general attack strategies and upper bounds for their complexity. Besides the vulnerabilities against differential and algebraic cryptanalysis, Bos et al. in [4] showed that secret keys of existing white-box implementations can be extracted by observing the addresses which are accessed during the execution if the external encodings are known to the adversary. The underlying so-called Differential Computational Analysis (DCA) applies the concept of Differential Power Analysis (DPA) [15] on eavesdropped address bits.

A first white-box implementation in hardware has been proposed for the NOEKEON cipher in [6,8] using 1-bit linear nibble encodings (i.e. masking with deterministic masks).

Our Contribution: In this work we propose a white-box implementation of AES dedicated to reconfigurable hardware. Although the white-box implementation of Chow et al. initially was proposed for software implementations, we show that the implementation can be mapped to existing reconfigurable hardware architectures. Note that only recent generations of reconfigurable hardware devices provide adequate amounts of resources to cope with the large memory requirements of white-box implementations.

For this hardware implementation we next examine the vulnerability to SCA attacks assuming a gray-box adversary model. These results, obtained from an FPGA platform extend the observation by Bos et al. (in [4]). We show that SCA attacks such as classical DPA can reveal the secrets in hardware implementations applying white-box cryptography even in gray-box settings.

Finally, we perform a thorough mathematical investigation and analysis of the construction of look-up tables used in white-box cryptography. We explain and verify the reason behind the success of such (DCA and DPA) attacks what has not been addressed in the seminal work of Bos et al.. Our results give a better understanding of the mathematical foundations of these attacks which

can pave the way for improved future white-box designs and implementations that are resistant against such analyses and threats.

Outline: The remainder of this article is organized as follows: Sect. 2 introduces the basic concept of white-box cryptography and gives a detailed explanation of the white-box implementation of Chow et al. including design and construction approach and known attacks and vulnerabilities. The process of transforming this white-box implementation into a hardware architecture (realized on an FPGA) is described in Sect. 3. In Sect. 4 we deal with gray-box adversary model and SCA attacks. We recap the concept of DCA and pinpoint the source of leakage of the given AES white-box implementation before we conclude in Sect. 5.

2 Background

This section introduces the basic concept of white-box cryptography and gives a detailed description of the seminal AES white-box implementation of Chow et al.

2.1 White-Box Cryptography

Cryptographic algorithms are designed to enable a secure communication even in the presence of an attacker. Nowadays, cryptographers differentiates between three common attacker models which try to estimate and model the capabilities of an adversary. Usually, modern cryptographic algorithms and their implementations are analyzed within such attacker models in order to deduce and estimate their security.

The traditional security and attacker model is the so-called black-box model which assumes a trusted execution environment and secure communication endpoints. In this model, cryptographic implementations are considered as black-box where an adversary can only observe the input and output behavior.

Since the development and deployment of embedded systems for security purposes the black-box model has been superseded by the gray-box model. This model includes the black-box settings but in addition assumes some expanded capabilities of a possible attacker. Cryptographic implementations are no longer considered as black-box but instead an adversary has limited access to the implementation internals which can be used to break the implementation. Note that gray-box attacks (e.g., SCA attacks) usually focus and target cryptographic implementations rather than cryptographic algorithms which still should be secure under the assumption of the black-box model.

However, another attacker model called white-box model has been introduced in particular for software implementations of cryptographic algorithms. For this model, the capabilities of an adversary are virtually unlimited since the attacker is assumed to have full control over the implementation and its execution environment. Aim of any implementation considered to be secure under the white-box model is to behave as a virtual black-box to any kind of attacker

such that even a white-box attacker should not have any additional advantage over black-box attackers. The ideal white-box implementation would consists of a single look-up table mapping a plaintext to its specific ciphertext already including a (hidden) secret key. Obviously this is impractical for modern ciphers with block and key sizes of 128 bits or more. An alternative approach is to transform the cryptographic primitive into a functionally-equivalent implementation using a series of smaller look-up tables. In a further step, secret and invertible encodings are applied to each look-up table individually in order to protect and hide secret key materials.

In general, the strategy for the design of white-box implementations of a round-based symmetric block cipher can be depicted as:

$$\underbrace{(\mathsf{f}^{(r+1)})^{-1} \circ \mathsf{E}^{r} \circ \mathsf{f}^{r}}_{table} \circ \cdots \circ \underbrace{(\mathsf{f}^{(3)})^{-1} \circ \mathsf{E}^{2} \circ \mathsf{f}^{2}}_{table} \circ \underbrace{(\mathsf{f}^{(2)})^{-1} \circ \mathsf{E}^{1} \circ \mathsf{f}^{1}}_{table}$$
$$= (\mathsf{f}^{(r+1)})^{-1} \circ \mathsf{E}^{r} \circ \cdots \circ \mathsf{E}^{2} \circ \mathsf{E}^{1} \circ \mathsf{f}^{1} = (\mathsf{f}^{(r+1)})^{-1} \circ \mathsf{E}_{K} \circ \mathsf{f}^{1},$$

where $\mathsf{E}^{i \in \{1 \dots r\}}$ is a single round instance of the block cipher and f^1 respectively $(\mathsf{f}^{r+1})^{-1}$ are considered as external input and output encoding of the white-box implementation (in order to prevent Code Lifting attacks [11]).

The white-box model has initially been proposed by Chow et al. [9] in 2002 when focusing on a fixed key implementation of the DES algorithm, and shortly afterwards a white-box implementation of the AES algorithm was presented [10]. In the following, we first introduce this seminal AES white-box implementation and discuss the design principles and known attacks and vulnerabilities under the white-box model before we show how to implement this design in hardware.

2.2 White-Box Implementation of AES

The architecture presented in [10] is a fixed key implementation with a fully unrolled design merging the atomic operations into a series of look-up tables. Basic design goals of this construction are to hide the key and algorithm structure through implementing the algorithm as a network of randomized look-up tables. Each look-up table is encoded and protected individually using random linear and non-linear bijections. Since a detailed discussion of the design would exceed the scope of this work we refer the interested reader to [19] and restrict the discussion of the white-box implementation to its basic design principle and construction.

Design and Construction: The transformation of an unprotected AES implementation (independently of the used key size) into a white-box protected fixed key implementation according to the scheme of Chow et al. can be achieved in two phases: first, the AES algorithm has to be rewritten and translated as a series of look-up tables and second, secret but invertible encodings have to be applied to all look-up tables in order to build a white-box implementation.

The following section will describe this process exemplary for the case of AES-128 as presented in [10], but again subdividing each phase into two steps.

In the following, we use the lower-case letter x for single bytes of the intermediate round state, $\hat{k}$ for a single byte of a round key, a raising index ^{r} for the current round and lowering indices $_{(i,j)}$ for the current byte position in the state matrix, where i denotes the row index and j the column index. Functions are represented with sans serif fonts. The AES S-box is denoted with $\mathsf{S}(.)$ and the matrix of the *MixColumns* operation is denoted by MC.

Step 1: Partial Evaluation. In the first step, the S-box computation is combined with the preceding addition of the round key. Merging both operations yields into a single look-up table defined as T-box:

$$\mathsf{T}^{r}_{i,j}(x) = \mathsf{S}(x \oplus \hat{k}^{r}_{i,j}) \qquad \text{for } 0 \leq i,j \leq 3 \text{ and } 1 \leq r \leq 9$$

$$\mathsf{T}^{10}_{i,j}(x) = \mathsf{S}(x \oplus \hat{k}^{10}_{i,j-i}) \oplus \hat{k}^{11}_{i,j} \qquad \text{for } 0 \leq i,j \leq 3$$

This step results in 160 different key-dependent T-boxes. It should be noted, that the T-boxes of the last round incorporate two bytes of two different round keys. This is due to the missing *MixColumns* operation and the final post-whitening key addition.

Step 2: Matrix Partitioning. A well-known implementation technique for the *MixColumns* operation is to decompose it into four different 8×32-bit look-up tables using the matrix partitioning strategy. Eventually, four 32-bit table outputs are added, resulting in the original *MixColumns* transformation. Applying this approach to our previously constructed T-boxes gives us a new set of different TMC tables, where MC_i denotes the i-th column of the MC matrix:

$$\mathsf{TMC}^{r}_{i,j}(x) = \mathsf{MC}_i \circ \mathsf{T}^{r}_{i,j}(x) \qquad \text{for } 0 \leq i,j \leq 3 \text{ and } 1 \leq r \leq 9$$

Finally, this results in 144 different 8×32-bit TMC look-up tables and additionally 16 different 8×8-bit T-boxes for the last round. Since all look-up tables comprise a small portion of the secret key, they have to be protected against attackers aiming at extracting the secret. For a better illustration, the key-dependent tables can be seen as miniature block ciphers that have to be enhanced by well-known techniques such as *diffusion* and *confusion* for protection purposes. Before applying randomly chosen invertible non-linear white-box encodings to the key-dependent tables in order to achieve *confusion*, *diffusion* is achieved through the application of linear transformations[1] called mixing bijections.

Step 3: Mixing Bijections. To add diffusion to each key-dependent table, two different linear transformations are necessary: an 8×8-bit linear transformations

[1] Note that originally affine and non-affine transformations were considered. However, since the constant of any affine transformation can be combined with the non-affine mapping, this eventually behaves as linear transformations.

$\mathsf{L}^r_{i,j}$ is inserted before $\mathsf{TMC}^r_{i,j}$, and a 32×32-bit transformation R^r_i is applied afterwards. In order to cancel out the effect of the transformation R^r_i after the addition of the TMC output values, another untwist table is introduced after each TMC table. This untwist table takes care of canceling the effect of the transformation R^r_i and applying new 8×8-bit transformations $(\mathsf{L}^{r+1}_{i,j})^{-1}$ to keep the encryption process consistent during all rounds. These transformations can be found by randomly creating linear matrices and checking for invertibility.

Step 4: Nibble Encodings. Eventually, non-linear white-box encodings are applied to all table inputs and outputs. For the sake of efficiency, concatenation of 4-bit nibble encodings were chosen rather than 8-bit byte encodings. Since these non-linear encodings avoid linear operation over the TMC table outputs, dedicated tables for the XOR operations have to be introduced. These nibble encodings can be found by constructing random 4-bit permutations. All in all, this design strategy results in five different look-up tables that are defined as follows:

$\mathcal{L}$-Ia:	$\mathsf{N}_{out} \circ \mathsf{R}^1_i \circ \mathsf{TMC}^1_{i,j} \circ (\mathsf{F}_{i,j})^{-1}$	$(8 \times 32\text{-bit})$
$\mathcal{L}$-Ib:	$\mathsf{G}_{i,j} \circ \mathsf{T}^{10}_{i,j} \circ \mathsf{L}^{10}_{i,j} \circ (\mathsf{N}_{in})^{-1}$	$(8 \times 8\text{-bit})$
$\mathcal{L}$-II:	$\mathsf{N}_{out} \circ \mathsf{R}^r_i \circ \mathsf{TMC}^r_{i,j} \circ \mathsf{L}^r_{i,j} \circ (\mathsf{N}_{in})^{-1}$	$(8 \times 32\text{-bit})$
$\mathcal{L}$-III:	$\mathsf{N}_{out} \circ (\mathsf{L}^{r+1}_{i,j})^{-1} \circ (\mathsf{R}^r_i)^{-1} \circ (\mathsf{N}_{in})^{-1}$	$(8 \times 32\text{-bit})$
$\mathcal{L}$-IV:	$\mathsf{N}_{in} \circ \mathcal{L}_{\oplus} \circ (\mathsf{N}_{out})^{-1}$	$(8 \times 4\text{-bit})$

Combining these tables in their designated way (a single round is depicted in Fig. 1) results in an encoded fixed-key white-box AES instantiation

$$\mathsf{AES}'_K = \mathsf{G} \circ \mathsf{AES}_K \circ \mathsf{F}^{-1},$$

where F^{-1} and G are responsible for external input and output encodings respectively.

Known Attacks and Vulnerabilities: Below we briefly outline the known attacks and vulnerabilities of the above presented white-box AES implementation. Some of the threats were already considered during its design. For those, we additionally explain how the attacks were targeted and how the countermeasures were integrated.

Code Lifting Attacks. Since the secret key is hidden and integrated into the white-box implementation, the goal of an attacker is obviously to extract the secret key. However, such fixed-key white-box implementations suffer from another kind of threat where an attacker is not interested in extracting the secret key but instead cloning the entire white-box implementation in order to use it at another place. This threat is known as "'Code Lifting"' where the entire white-box application is seen as a single key that is cloned and misused by an attacker to encrypt and decrypt data without being in possession of the

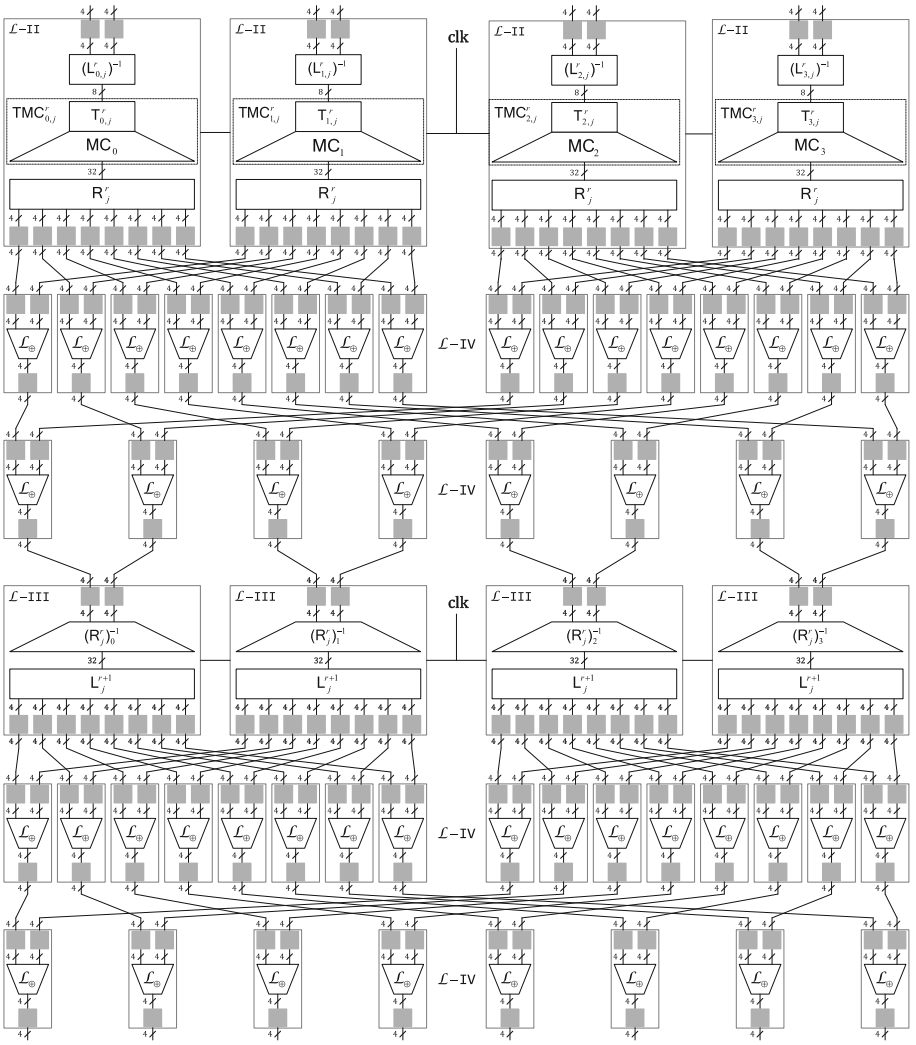


Fig. 1. White-box implementation of a quarter AES round

secret key. To avoid such kind of attacks, external encodings (F and G) are introduced, turning an white-box implementation E_K into an obfuscated encryption function $\mathsf{E}'_K = \mathsf{G} \circ \mathsf{E}_K \circ \mathsf{F}^{-1}$ with hidden external encodings. By pushing the white-box implementation boundaries, the attacker is no longer able to misuse the white-box implementation as long as the external encodings are unknown.

White-Box Inversion. Besides cloning the white-box implementation through Code Lifting, inverting the encryption (or decryption) function is another practical issue in particular for white-box implementations of AES. Since the entire

algorithm is implemented through look-up tables, any white-box attacker would be able to extract the tables and compute the inverses of all rounds. This allows to turn any implemented encryption (respectively decryption) function into an decryption (respectively encryption) without knowing the secret key. In fact, this issue cannot be prevented but mitigated by external encodings since it prevents the attacker to use the inverted function in a meaningful way. In particular the application of non-invertible external encodings can prevent the inversion of white-box implementations ensuring the property of *one-wayness*.

Stripping of Non-Linear Encodings. A first algebraic analysis of the above-explained white-box AES implementation has been presented by Billet et al. [3] which revealed serious vulnerabilities of this design approach by stripping of the non-linear encodings of the look-up tables and allowing a white-box attacker to efficiently extract the embedded secret key. Later, Michiels et al. [17] generalized this attack for any cipher following the substitution-linear transformation (SLT) approach. In general, Billet et al.'s approach considers a quarter of the AES round function (depicted in Fig. 1) as a single 32×32-bit function rather than a decomposition into a series of look-up tables. Following this strategy, the influence of the mixing bijection R_i^r and any other internal (non-linear) encoding are canceled out.

It was observed, that with moderate computational effort, the non-linear encodings at the beginning and end of each quarter AES round can be removed, so that only some (unknown) affine transformation will remain. Applying this technique to three subsequent rounds, thus removing the non-linear encodings up to an affine part, the secret key eventually can be retrieved with a complexity of at maximum 2^{30} (cf. [3]). Note, however, that this attack is only possible in the setting of white-box adversaries, since an attacker needs to have full access to the tables and control over their inputs and outputs.

3 FPGA Implementation

This section briefly introduces modern reconfigurable hardware architectures exemplary considering Xilinx FPGAs and describes necessary hardware resources to implement white-box cryptography efficiently in reconfigurable hardware. Afterwards, the approach of transforming the white-box AES implementation of Chow et al. into an efficient hardware architecture for recent Xilinx Kintex-7 FPGAs is outlined. Finally, we give performance and implementation results on the area and throughput efficiency of the proposed architecture.

3.1 Hardware Resources

Modern FPGAs consist of a sea of general-purpose logic resources that can implement arbitrary circuits of Boolean functions using small look-up tables. The logic resources are arranged in an extremely regular array-like structure and enhanced by special purpose units e.g., Digital Signal Processors (DSPs) or

Block Memories (BRAMs). The reconfigurable devices are programmed using a configuration file called *bitstream* that contains all configuration information for implemented hardware resources, i.e., the programmable interconnections, the general purpose logic and the special purpose resources.

General Purpose Logic Resources: Xilinx decided to cluster several general purpose logic resources as Configurable Logic Blocks (CLBs) and arrange them in a grid-like structure of rows and columns. Starting with the Virtex-5 family of Xilinx devices, each CLB constitutes two slices each equipped with four 6-input Look-Up Tables (LUTs) and eight adjacent Flip-Flops (FFs) to implement any circuit of Boolean functions. Starting with the newer 7-series devices, only two different types of slices (Slice-L and Slice-M) were implemented which only differ in capabilities of using LUTs as distributed memory instead of function generators. Both, Slice-L and Slice-M instances, provide some wide multiplexers that allow to connect the outputs of the LUTs in order to implement any 8×1-bit Boolean function efficiently into a single slice.

Dedicated Block Memory Resources: Besides general purpose logic that can also serve as (distributed) memory, modern FPGAs provide larger amounts of data storage in terms of dedicated BRAMs. These flexible, low-power memory units can be configured by the user and provide between 16-Kbits to 32-Kbits accessible in single or dual port mode (additionally, 2-Kbits respectively 4-Kbits memory for parity check purposes are available). In dual port mode, two fully independent ports providing read and write access (even with different clocks) can be used to access or manipulate data that is stored in memory. In addition, each BRAM can be configured individually and used in different configurations considering port width and memory depth, ranging from 32K $\times$ 1-bit to 1K $\times$ 32-bit entries.

3.2 White-Box Architecture in Hardware

White-box cryptography was initially proposed to protect software implementations. In this context we like to remark that bitstream configuration files of FPGA designs are digital binary files that are stored in external memory (that are accessible for an attacker) and thus exposed to very similar threats. Further, the basic idea of white-box implementations is to transform a cryptographic implementation into a series of look-up tables. This perfectly fits the regular structure of FPGAs implementing arrays of look-up tables with programmable interconnections. Hence we can conclude that FPGAs seem to be a very good fit for cryptographic white-box implementations in hardware.

However, since every individual look-up table of the white-box implementation is different (due to different round keys and randomly chosen encodings), we cannot implement any area-efficient round-based or serialized architecture of the AES algorithm nor reuse any of the look-up tables. Instead, we have to implement an entirely unrolled implementation with every round instantiated

separately. Due to the application of BRAM primitives, which have a latency of a single clock cycle, this causes an initial latency of 19 clock cycles (due to 19 stages of 16 parallel look-up tables in the proposed white-box implementation) but in order to increase the throughput it is possible to operate the encryption architecture in a pipeline fashion providing ciphertexts at each clock cycle (after the initial latency).

Mapping Tables into CLBs: Besides the implementation of the key depending TMC-Tables and T-boxes, the encoded look-up tables to perform the XOR operations consume a large part of the required storage. Although modern FPGAs provide large amounts of general purpose data storage in terms of BRAM, implementing all look-up tables using these dedicated memory primitives is still not feasible. Therefore, some tables have to be transferred to the general purpose logic in order to fit the design into an FPGA. Since any 8×1-bit Boolean function can be implemented efficiently into a single slice and each XOR operation and its corresponding look-up table can be decomposed into four different 8×1-bit functions, it is a natural choice to implement these tables in general purpose logic. In total, each XOR-table can be implemented using four slices equipped with 4 LUTs each, thus in total 16 LUT instances are required (this equals 1024-bit memory). Fortunately, the last round can do without XOR operations, so we only have to implement these tables for 9 rounds. As depicted in Fig. 1, a quarter round of the AES white-box implementation implements 48 XOR-tables which results in 192 tables per full round and 1728 tables in total.

Mapping Tables into Block Memory: The remaining look-up tables can be implemented in BRAM primitives. Most of the tables, except for the T-boxes of the last round, 8×32-bit functions are implemented which requires 8192-bit of memory. Since we can use the BRAM in dual port mode, two tables can be implemented in a single BRAM which allows us to entirely use the 16-Kbit BRAMs resulting in a very dense and efficient implementation. In total, as depicted in Fig. 1, 8 different look-up tables with 32-bit output values are implemented in a quarter AES round, thus 32 tables are necessary to build a full round function (except for the last round). In total, 176 different such tables have to be instantiated along with 16 different 8×8-bit T-boxes for the last round. Note, that all BRAM tables have a similar shape except for the first and last round.

3.3 Performance Evaluation

Table 1 provides the memory consumption of our white-box implementation of AES-128 broken down to different look-up table types and their implementation size (resources and memory). In total, 536KB of memory are required to implement this white-box implementation on an FPGA, whereby 41 % of the memory is required for tables of type $\mathcal{L}$-IV implemented in logic and the remaining 59 % of memory is necessary to store tables of type $\mathcal{L}$-I to $\mathcal{L}$-III in BRAMs.

Table 1. Area and memory consumption of different table types

Look-up tables			Resources		Memory
No.	Type	Size	LUT	BRAM	Byte
16	$\mathcal{L}$-Ia	(8 × 32-bit)	-	8	16 384
16	$\mathcal{L}$-Ib	(8 × 8-bit)	-	8	4 096
144	$\mathcal{L}$-II	(8 × 32-bit)	-	72	147 456
144	$\mathcal{L}$-III	(8 × 32-bit)	-	72	147 456
1728	$\mathcal{L}$-IV	(8 × 4-bit)	27 648	-	221 184
Total			27 648	160	536 576
Utilization (for XC7K160T)			28 %	46 %	40 %

As mentioned before, the design has an initial latency of 19 clock cycles introduced by the BRAM stages. If operated in pipelined mode, this architecture can return one ciphertext per clock cycle after the initial 19 clock cycles. Due to the pipelined architecture and small critical paths, the entire design can operate at a maximum frequency of 100 MHz, resulting in a final throughput of 12.8 Gbit/s. Implementing this on a recent Xilinx Kintex-7 XC7K160T, this design occupies roughly 28 % of the available slices and 46 % of provided BRAM ressources.

4 Side-Channel Analysis

4.1 Differential Computational Analysis Attack

Recently, Bos et al. introduced a new analysis methodology for cryptographic white-box implementations in [4] which requires neither knowledge or possession of the implemented and used look-up tables nor reverse-engineering the tables during the attack process. The following section briefly introduces the methodology of the DCA attack in order to extract secret keys from unknown white-box implementations.

Methodology: DCA primarily targets software-based white-box implementations. In order to successfully perform a key-recovery attack the following two conditions have to be fulfilled:

1. The attacker is able to execute the white-box implementation several times, with different (randomly chosen) plaintexts.
2. Either input- or output external encodings are known to the attacker.

In particular the second requirement is of major importance since it already implies that this attack can be prevented if external encodings are applied and kept secret. However, in practice, at least one encoding (either the initial encoding or the final decoding) usually is known by the user in order to allow a

meaningful application of the encryption (or decryption) function. If both aforementioned conditions are fulfilled, assuming that the underlying cryptographic algorithm is known to the attacker, the following three steps can be followed to perform a DCA attack.

Step 1: Record Multiple Measurements. It is assumed, that the adversary can execute the white-box implementation in a fully controlled environment. During multiple execution of the encryption algorithm with randomly chosen plaintexts, all accessed memory addresses and any data written to or read from memory are recorded.

Step 2: Conversion to Ideal Traces. A certain type of information is extracted from the recorded data. Common examples of promising information are data read from memory (corresponding to the look-up table outputs), data written to stack (intermediate values of the encryption process) or parts of memory addresses (corresponding to inputs of the look-up tables). The extracted data is converted to a format that can be used by common DPA tools. The authors proposed to serialize the recorded data into a binary string and append the results according to their temporal occurrence. This final binary string is handled as a kind of side-channel trace that we denote to as Ideal Trace since it refers to the result of a fully noise-free probing process.

Step 3: Perform DPA Attack. Following the concept of classical DPA, by guessing a key byte k^* and knowing the corresponding plaintext bytes p, the output bits of the S-box, i.e., $\mathsf{S}(p \oplus k^*)$, are predicted. Using these models (8 for each key byte) DPA attacks are performed on the Ideal Traces to distinguish the correct key guess amongst the others.

Although the authors of [4] reported successful key recoveries, the reason behind such a success has not be clearly stated. Below we first address our observations from an SCA adversary point of view, and later deal with the leakage source.

4.2 Differential Power Analysis Attack

In this scenario we supposed a gray-box adversary model, where the underlying cryptographic algorithm (e.g., AES) is known, but no information about the type of the implementation and its structure (e.g., white-box or ordinary design) is known to the attacker. Further, we suppose that there is no external encoding in the design, e.g., the gray-box seen by the attacker performs standard AES encryption (or decryption). However, the adversary is able to observe side-channel information (e.g., power consumption) of the implementation while it is operated.

Measurement Setup. We made use of a SAKURA-X FPGA board [1] equipped with a Kintex-7 XC7K160T FPGA to practically examine the vulnerability of our white-box design with respect to such an SCA adversary. By means of a

digital oscilloscope, the side-channel traces have been collected by measuring the voltage drop over a 1 Ω resistor in the V_{dd} path of the FPGA during the operation of the design. The sampling was performed at a rate of 500 MS/s and a bandwidth limit of 20 MHz while the design was running at a stable, jitter-free, but low clock frequency of 3 MHz to mitigate the noise. During the measurement phase, our hardware implementation of white-box AES was provided by fully random plaintexts. A sample power trace, where the rounds (19 clock cycles) are clearly distinguishable, is shown in Fig. 2.

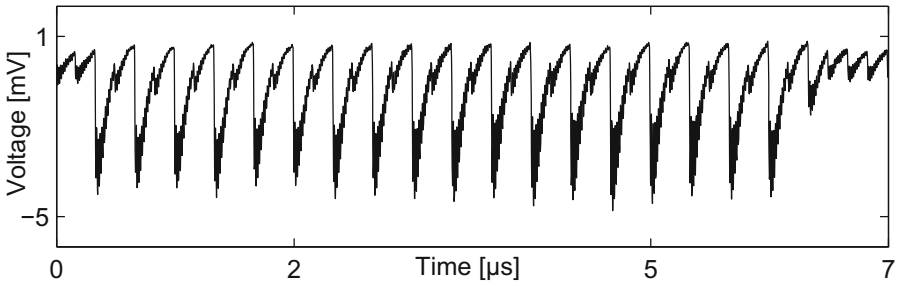


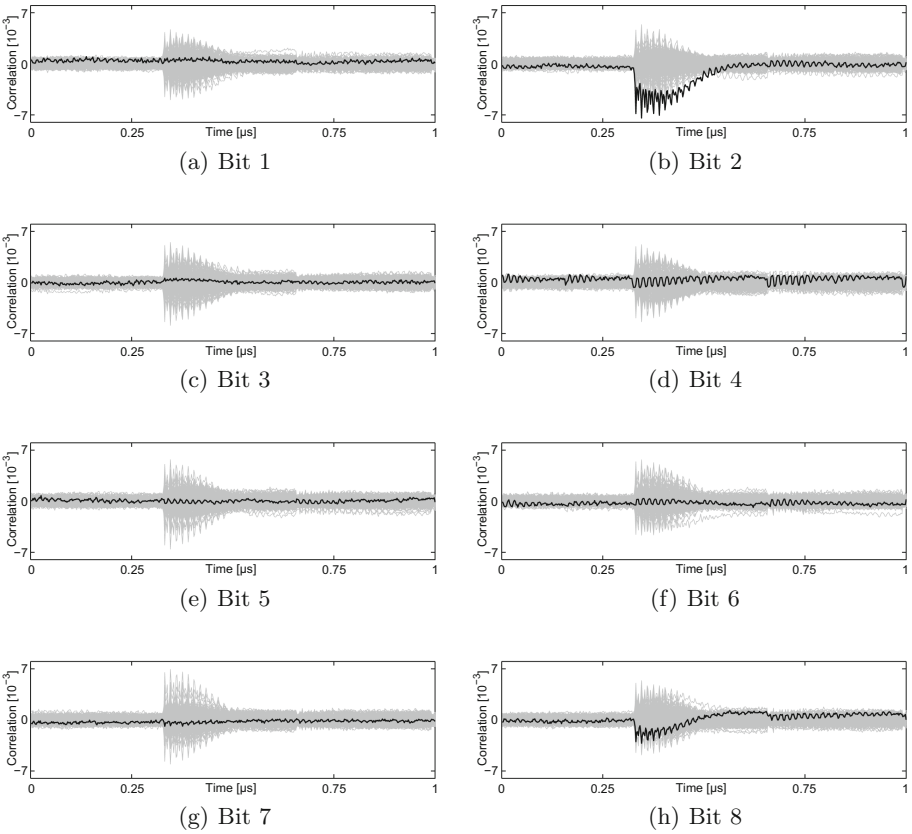
Fig. 2. A sample power trace.

Evaluation. We have collected 10 million power traces of encryptions while the plaintexts were selected randomly. In fact, we have applied several different variants of power analysis attacks including CPA [5], DPA [15] and collision ones [18] with different hypothetical models. The best result has been achieved by means of the classical DPA, which is the same as CPA with single-bit power model. Similar to the case of DCA, for each key byte candidate k^* the output bits of the S-box at the first round, i.e., $\mathsf{S}(p \oplus k^*)$, have been predicted and correlated to the power traces. The results of such 8 different CPA attacks on each bit of one of the S-box outputs are shown in Fig. 3. As shown by the graphics, only one of the attacks (bit 2) is able to recover the secret. We have performed the same attacks on all 16 S-boxes of the first round. Although the attacks on different S-boxes did not show identical results, at least one of the output bits of each S-box led to a successful key recovery, hence full 128-bit key could be recovered.

We would like to note that DCA [4] is indeed a CPA with single-bit power model, assuming the identity function as the actual leakage model of the device and noise-free measurements. Hence, we have shown that the attack is still feasible in case of imperfect (i.e., noisy) measurements and a more complex side-channel leakage function.

4.3 Mathematical Foundations

In order to discuss about the reason behind such a leakage, we first need to give the following definitions.



(a) Bit 1 (b) Bit 2

(c) Bit 3 (d) Bit 4

(e) Bit 5 (f) Bit 6

(g) Bit 7 (h) Bit 8

Fig. 3. CPA results, S-box output bit model, 10 million traces

Definition 1. *Let* $x =< x_1, ..., x_n >$, $\omega =< \omega_1, ..., \omega_n >$ *be elements of* $\{0,1\}^n$ *and* $x \cdot \omega = x_1\omega_1 \oplus ... \oplus x_n\omega_n$. *Let* $f(x)$ *be a Boolean function of* n *variables. Then the Walsh transform of the function* $f(x)$ *is a real valued function over* $\{0,1\}^n$ *that can be defined as* $W_f(\omega) = \sum_{x \in \{0,1\}^n} (-1)^{f(x) \oplus x \cdot \omega}$.

Definition 2. *Iff the Walsh transform* W_f *of a Boolean function* $f(x_1, ..., x_n)$ *satisfies* $W_f(\omega) = 0$, *for* $0 \leq HW(\omega) \leq m$, *it is called a balanced m-th order correlation immune (CI) function or an m-resilient function, where HW stands for Hamming weight.*

For the sake of simplicity, we consider Fig. 4 as one of the 8-to-32 bit $\mathcal{L}$-`Ia` look-up tables used at the first round of our white-box implementation. As stated before, it is supposed that no external encodings exist in the design (or they are known to the adversary), hence we did not draw them in the figure. Let us denote the output of the S-box by x and the combination of MC and linear encoding R and non-linear 4-to-4 bit encodings by 32 Boolean functions $f_{i \in \{1,...,32\}}(x) : \{0,1\}^8 \rightarrow \{0,1\}$. The results of CPA and DCA indicate that at least one of these

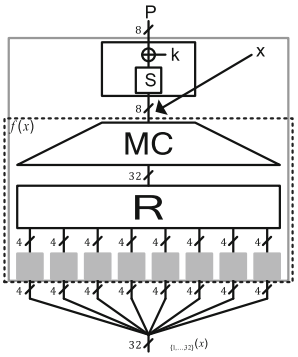


Fig. 4. Detailed representation of an 8×32 look-up table at the first round of our white-box design

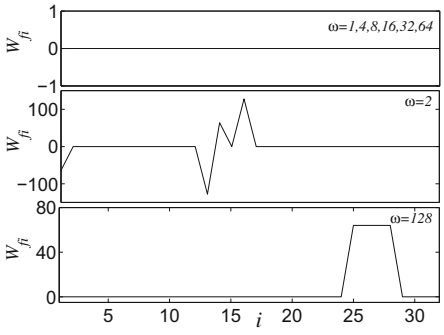


Fig. 5. Walsh transforms for all 32 functions $f_{i\in\{1,\ldots,32\}}(\cdot)$ with $HW(\omega) = 1$.

functions $f_i(\cdot)$ is not first-order correlation immune. In order to investigate this, we calculated the Walsh transform of all these functions for all $\omega \in \{0,1\}^8$. The results for 8 cases, where $HW(\omega) = 1$, are shown in Fig. 5.

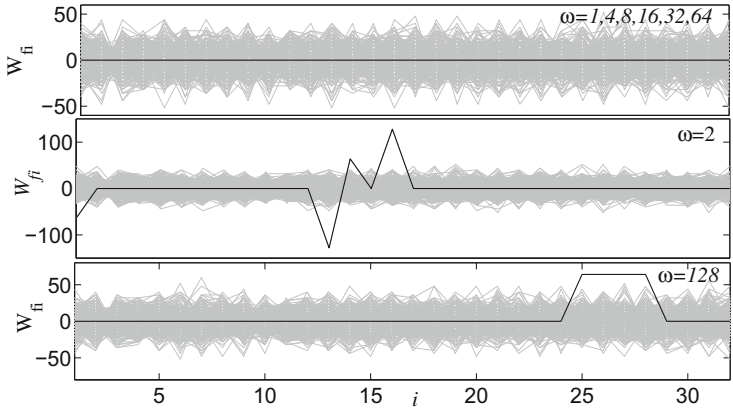


Fig. 6. Walsh transforms for all 32 functions $f_{i\in\{1,\ldots,32\}}(\cdot)$ with $HW(\omega) = 1$ for all key candidates $k^* \in \{0,1\}^8$.

As shown by the graphics, Walsh transform of a couple of functions for two particular ω show an extreme imbalance. However, this fact does not guarantee that a CPA or DPA leads to a successful key recovery. To clarify this fact, we suppose that the linear encoding R and non-linear 4-to-4 bit encodings are unknown, and for each key candidate k^* we derive $f_{i\in\{1,\ldots,32\}}(x)$ by 32-bit output of $\mathcal{L}\text{-}\mathtt{Ia}\left(p = \mathsf{S}^{-1}(x) \oplus k^*\right)$. For each key candidate $k^* \in \{0,1\}^8$ we again calculated the Walsh transforms for all $\omega \in \{0,1\}^8$. Figure 6 represents the results of

each $\omega; HW(\omega) = 1$ over all key candidates. As shown by the figures, for $\omega = 2$ the extreme imbalance of some functions $f_{i\in\{1,\ldots,32\}}(\cdot)$ for the correct key can be detected amongst that for other key candidates. This indeed justifies why DCA and CPA led to successful key recoveries as this observation perfectly fits to the result of CPA on the same key byte (as shown in Fig. 3), where similarly only second bit of the S-box output (compatible with $\omega = 2$) led to successful key recovery. It is noteworthy that we have similarly examined all other look-up tables of the first cipher round, and for each of them the Walsh transform of at least one $\omega; HW(\omega) = 1$ for the correct key showed extremely high imbalance (compared to that for other key candidates). We should stress that all linear and non-linear encodings used in our design have been randomly generated as stated in Step 3 and Step 4 of Sect. 2.2.

4.4 How to Avoid Such Attacks

At the first glance, it can be concluded that if any imbalances is avoided in functions $f_{i\in\{1,\ldots,32\}}(\cdot)$, i.e., all f_i to be first-order correlation immune, DPA and DCA can be avoided. However, it should be noted that such a correlation immunity is valid only in case of classical DPA. In other words, if any of the functions f_i has an extremely high imbalance for any $\omega \in \{0,1\}^8$, that makes it recognizable compared to other key candidates, there exists an attack which can recover the correct key. Such an attack can make use of a power model (or distinguisher) corresponding to that ω. Alternatively, those power analysis attacks which consider the distribution of the leakages, e.g., Mutual Information Analysis [12] which relaxes the power model, can be applied.

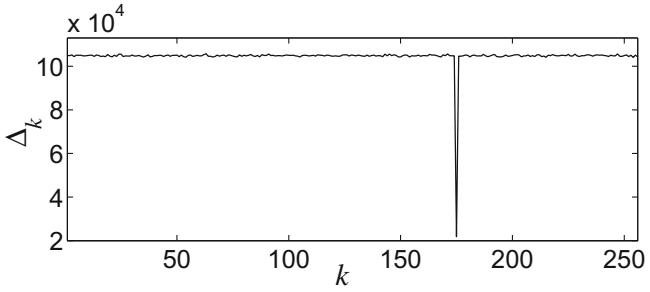


Fig. 7. Sum of all imbalances Δ_k for all key candidates.

In contrary, if many of the functions f_i are m-correlation immune (for any arbitrary m), this opens another door to recover the key. Suppose that for all key candidates k^* and for all ω we calculated the Walsh transforms W_{f_i}. If we sum up all the imbalances for each key candidate as

$$\Delta_{k\in\{0,1\}^8} = \sum_{\forall\omega\in\{0,1\}^8} \sum_{i=1,\ldots,32} \left| W_{f_i}(\omega) \right| \; ; \; k^* = k,$$

the Δ_k for the correct key candidate might be distinguishable (though minimum). In case of our design (the same look-up table which have been considered above), Fig. 7 shows Δ_k for all key candidates, where the correct key is obviously distinguishable. In fact, these results indicate that the linear and non-linear encodings cannot be arbitrary (randomly) selected. Otherwise, the key can be easily revealed by the above explained procedure. This raises a question as what should be the characteristics of such random encodings in such a way that these attacks are not applicable. At least, it can be said that $\forall\omega$ the distribution of Walsh transforms of all f_i should be not distinguishable from that of other key candidates. But how to define the corresponding characteristics to fulfill such a property is considered as future works.

5 Conclusion

In this paper, we presented the first white-box implementation of AES realized in reconfigurable hardware. Assuming a gray-box adversary model, we have practically examined the resistance of our architecture against side-channel attacks. Unfortunately, we were able to successfully perform attacks using classical DPA. However, our observations approve previous results on software-based white-box implementations and extend these results to hardware implementations and physical side-channel attacks. Finally, we provide a to-date missing thorough mathematical analysis of the underlying reasons that enable attacks on such white-box implementations even assuming a gray-box model in case of a lack of unknown external encodings.

Directions for future works include (i) specifying the requirements of linear and non-linear encodings in such a way that the tables cannot be analyzed through their imbalances and (ii) developing designs of new white-box implementations to provide resistance against side-channel attacks. In practice, a conceivable approach to avoid vulnerabilities of white-box implementations in a gray-box adversary model might be a dynamic update of intermediate encodings. In particular for reconfigurable devices, which offer partial reconfiguration abilities, this might be an interesting approach to make side-channel attacks practically infeasible.

Acknowledgment. The authors would like to thank Gregor Leander from Ruhr University Bochum (Germany) for helpful discussions and his comments on the application of Walsh transform.

References

1. Side-channel AttacK User Reference Architecture. http://satoh.cs.uec.ac.jp/SAKURA/index.html
2. Baek, C.H., Cheon, J.H., Hong, H.: Analytic toolbox for white-box implementations: limitation and perspectives. IACR Cryptol. ePrint Arch. **2014**, 688 (2014)

3. Billet, O., Gilbert, H., Ech-Chatbi, C.: Cryptanalysis of a white box AES implementation. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 227–240. Springer, Heidelberg (2004)
4. Bos, J.W., Hubain, C., Michiels, W., Teuwen, P.: Differential computation analysis: hiding your white-box designs is not enough. IACR Cryptol. ePrint Arch. **2015**, 753 (2015)
5. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004)
6. Bringer, J., Chabanne, H., Danger, J.: Protecting the NOEKEON cipher against SCARE attacks in fpgas by using dynamic implementations. In: Prasanna, V.K., Torres, L., Cumplido, R. (eds.) 2009 Proceedings of the International Conference on Reconfigurable Computing and FPGAs, ReConFig 2009, Cancun, Quintana Roo, Mexico, pp. 9–11, pp. 183–188. IEEE Computer Society, December 2009
7. Bringer, J., Chabanne, H., Dottax, E.: White box cryptography: another attempt. IACR Cryptol. ePrint Arch. **2006**, 468 (2006)
8. Cherif, Z., Flament, F., Danger, J., Bhasin, S., Guilley, S., Chabanne, H.: Evaluation of white-box and grey-box noekeon implementations in FPGA. In: Prasanna, V.K., Becker, J., Cumplido, R. (eds.) 2010 Proceedings of the International Conference on Reconfigurable Computing and FPGAs, ReConFig 2010, Cancun, Quintana Roo, Mexico, pp. 13–15, pp. 310–315. IEEE Computer Society, December 2010
9. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: A white-box DES implementation for DRM applications. In: Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers, pp. 1–15 (2002)
10. Chow, S., Eisen, P.A., Johnson, H., van Oorschot, P.C.: White-box cryptography and an AES implementation. In: Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15–16, 2002, Revised Papers, pp. 250–270 (2002)
11. Delerablée, C., Lepoint, T., Paillier, P., Rivain, M.: White-box security notions for symmetric encryption schemes. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 247–264. Springer, Heidelberg (2014)
12. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008)
13. Goubin, L., Masereel, J.-M., Quisquater, M.: Cryptanalysis of white box DES implementations. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 278–295. Springer, Heidelberg (2007)
14. Karroumi, M.: Protecting white-box AES with dual ciphers. In: Information Security and Cryptology - ICISC 2010–13th International Conference, Seoul, Korea, December 1–3, 2010, Revised Selected Papers, pp. 278–291 (2010)
15. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
16. Lepoint, T., Rivain, M., Mulder, Y.D., Roelse, P., Preneel, B.: Two attacks on a white-box AES implementation. In: Selected Areas in Cryptography - SAC 2013–20th International Conference, Burnaby, BC, Canada, August 14–16, 2013, Revised Selected Papers, pp. 265–285 (2013)
17. Michiels, W., Gorissen, P., Hollmann, H.D.L.: Cryptanalysis of a generic class of white-box implementations. In: Avanzi, R.M., Keliher, L., Sica, F. (eds.) SAC 2008. LNCS, vol. 5381, pp. 414–428. Springer, Heidelberg (2009)

18. Moradi, A., Mischke, O., Eisenbarth, T.: Correlation-enhanced power analysis collision attack. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 125–139. Springer, Heidelberg (2010)
19. Muir, J.A.: A tutorial on white-box AES. IACR Cryptol. ePrint Arch. **2013**, 104 (2013)
20. Mulder, Y.D., Roelse, P., Preneel, B.: Cryptanalysis of the Xiao - Lai white-box AES implementation. In: Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15–16, 2012, Revised Selected Papers, pp. 34–49 (2012)
21. De Mulder, Y., Wyseur, B., Preneel, B.: Cryptanalysis of a perturbated white-box AES implementation. In: Gong, G., Gupta, K.C. (eds.) INDOCRYPT 2010. LNCS, vol. 6498, pp. 292–310. Springer, Heidelberg (2010)
22. Wyseur, B., Michiels, W., Gorissen, P., Preneel, B.: Cryptanalysis of white-box DES implementations with arbitrary external encodings. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 264–277. Springer, Heidelberg (2007)
23. Xiao, Y., Lai, X.: A secure implementation of white-box AES. In: 2009 2nd International Conference on Computer Science and its Applications (2009)