Hiding Higher-Order Side-Channel Leakage
– Randomizing Cryptographic Implementations in Reconfigurable Hardware –

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

[1] Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Bochum, Germany
pascal.sasdrich@rub.de, amir.moradi@rub.de

[2] University of Bremen & DFKI, Bremen, Germany
tim.gueneysu@uni-bremen.de

Abstract. First-order secure Threshold Implementations (TI) of symmetric cryptosystems provide provable security at a moderate overhead; yet attacks using higher-order statistical moments are still feasible. Cryptographic instances compliant to Higher-Order Threshold Implementation (HO-TI) can prevent such attacks, however, usually at unacceptable implementation costs. As an alternative concept we investigate in this work the idea of *dynamic hardware modification*, i.e., random changes and transformations of cryptographic implementations in order to render higher-order attacks on first-order TI impractical. In a first step, we present a generic methodology which can be applied to (almost) every cryptographic implementation. In order to investigate the effectiveness of our proposed strategy, we use an instantiation of our methodology that adapts ideas from White-Box Cryptography and applies this construction to a first-order secure TI. Further, we show that dynamically updating cryptographic implementations during operation provides the ability to avoid higher-order leakages to be practically exploitable.

1 Introduction

Side-channel analysis (SCA) uses information leakage by measuring physical device internals, e.g., timing [9], power consumption [10] or electromagnetic emanations [2], to extract cryptographic secrets. Modern side-channel countermeasures are classified either as *hiding* or *masking* [11]. While *hiding* countermeasures aim to decrease the Signal-to-Noise ratio (SNR) in order to hide information leakage in random noise, *masking* countermeasures tackle information leakage using secret sharing and multi-party computation techniques. The idea of Threshold implementation (TI) has been developed based on Boolean masking in particular to target hardware implementations [15]. However, the initial concept of TI was only suitable to counteract first-order side-channel leakages, still allowing attacks using higher-order statistical moments to successfully recover cryptographic secrets. Naturally, higher-order Threshold Implementations (HO-TI) have been proposed to solve this problem [4]. Despite, HO-TI might be limited

to uni-variate scenarios [18] as well as they come with increased time overhead and area demands due to the ever increasing number of minimum shares for higher-order protection. Therefore, combining first-order secure TI with *hiding* countermeasures to achieve (practical) higher-order protection might be an alternative solution.

Previous Work: Although the threat of side-channel attacks is well known, many cryptographic devices are vulnerable to side-channel analysis due to their static design and behavior which allows attacks based on statistical and differential analysis. Introducing dynamic behavior in terms of ever-changing and morphing implementations and circuits could help to overcome these problems. However, this is not a trivial task and poses big challenges to designers of cryptographic implementations in particular using static hardware devices. In recent years, several research into this direction has been performed and published but still existing solutions are at an early stage and have to face many difficulties. In 2008, Mentens et *al.* [12] introduced a first work for using dynamic reconfiguration of modern FPGAs as countermeasure against power and fault attacks. However, their solution had to struggle with slow reconfiguration times as well as too small complexity which still allowed efficient analysis and attacks. Besides, their solution specifically targeted reconfigurable hardware which provides dynamic reconfiguration features. Moradi and Mischke [13] examined the opportunities of using dual ciphers as alternative representations (in particular for AES) in order to achieve protection against side-channel attacks. Though, dual ciphers maintain structural properties of the original representation which again could be exploited using statistical analysis. Recently, Sasdrich et *al.* [19] proposed the application of affine equivalence representations of cryptographic S-boxes to change the cipher implementation dynamically during runtime. Although the complexity of this approach is quite high, it exploits very specific properties of the cryptographic components, so that this approach cannot be generically applied to cryptographic implementations.

Contribution: Our contribution in this work is twofold: First, we present a generic approach to change the representations of cryptographic implementations dynamically in order to introduce non-static behavior. Our methodology can be applied to (almost) every cryptographic implementation and circuit independent of the cryptographic algorithm or scheme. Our approach uses random substitution of basic elements along with random encoding of intermediate connections and offers high flexibility and scalability of attack complexities depending of the used level of abstraction and granularity of the underlying circuit. Second, we investigate and analyze a specific instantiation of our approach to randomize a TI. In particular, we are going to examine a first-order PRESENT TI as a case study which is implemented in reconfigurable hardware. The randomization of intermediate signals, in terms of random non-linear 4-bit encodings, is chosen dynamically during runtime and injected into each implemented look-up table in order to substitute them by different representations. In particular, this ap-

proach adapts ideas and techniques from the area of White-Box Cryptography (WBC), although we want to emphasize that we do not aim to achieve resistance against attacks in the white-box adversary model. Eventually, we conduct practical side-channel measurements for our case study. Using a leakage assessment methodology, we focus on effects of our countermeasure on higher-order statistical properties and moments and show that our approach can increase the protection against higher-order side-channel attacks from a practical point of view.

Outline: The remainder of this article is organized as follows: Section 2 summarizes and provides important background information on directed graphs, Threshold Implementations and White-Box Cryptography. In Section 3 we present a description of our generic approach to dynamically update and randomize cryptographic implementations which is applied in a case study in Section 4 using a specific instantiation based on a PRESENT TI and 4-bit non-linear encodings (as proposed for WBC). Section 5 provides side-channel evaluation results using power measurements and state-of-the-art leakage assessment methodologies. Eventually, our work concludes in Section 6.

2 Background

This section briefly introduces the theory of directed graphs before we recapitulate the background of Threshold Implementations (TI) and White-Box Cryptography (WBC).

2.1 Notations

We denote single-bit random variables using lower-case characters, bold ones for multi-bit vectors, bars for shared representations, lowering indices for elements within a vector and raising indices for elements of a shared vector.

Furthermore, let us denote any element $\boldsymbol{x} \in \mathbb{GF}(2^m)$ as vector of m single bit elements $\langle x_1, \ldots, x_m \rangle$. The shared representation $\bar{\boldsymbol{x}}$ of a vector $\boldsymbol{x}$ using Boolean masking with s shares is given as $\bar{\boldsymbol{x}} = (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^s)$, where

$$\boldsymbol{x} = \bigoplus_{i=1}^{s} \bar{\boldsymbol{x}} = \bigoplus_{i=1}^{s} \boldsymbol{x}^i = \bigoplus_{i=1}^{s} \langle x_1^i, \ldots, x_m^i \rangle.$$

Eventually, we denote functions using sans serif fonts and sets using calligraphic ones.

2.2 Directed Graphs

Directed Graphs (or digraphs) are use for many applications in order to abstractly model a certain problem and find according solutions. In general, a graph is a set of nodes that are connected by some edges. For a directed graph, the edge are provided with a certain direction.

Definition 1. *A **directed graph** or **digraph** is an ordered pair of sets $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where $\mathcal{V}$ is a set of vertices and $\mathcal{A}$ is a set of ordered pairs $a_{ij} = \langle v_i, v_j \rangle$ (called arrows or directed edges) with $v_i, v_j \in \mathcal{V}$.*

In particular, each vertex has a certain number of connected edges. Due to the direction of the edges, we can distinguish between edges that arrive at a vertex and edges that leave a vertex. The number of arriving edges is given by the in-degree of a node, whereas the number of leaving edges is given by the out-degree.

Definition 2. *In a directed graph, the **in-degree** $deg^+(v)$ and the **out-degree** $deg^-(v)$ of a vertex $v \in \mathcal{V}$ count the number of directed edges connecting to and from a vertex respectively. It holds, that $\sum_{v \in \mathcal{V}} deg^+(v) = \sum_{v \in \mathcal{V}} deg^-(v) = |\mathcal{A}|$.*

Eventually, every node (connected to a digraph) has to have at least one arriving or leaving edge. In case the node has an in-degree of zero, it is called source, since it only serves as starting point for several edges. Similarly, a node without any leaving edges is called sink, since it is only an ending point for some edges. In the following, we consider the source nodes as starting points of our directed graph, whereas the sink will be the final points.

2.3 Threshold Implementation

Threshold Implementation (TI) is a widely used technique to protect hardware devices against physical attacks. In particular, TI is based on Boolean masking and multi-party computation and provides provable security, even in the presence of glitches[3]. In general, any Threshold Implementation has to provide the following three properties:

Correctness: Given a vector $\bar{\boldsymbol{x}} = (\boldsymbol{x}^i, \ldots, \boldsymbol{x}^s)$ in its shared representation, we can compute any function $\mathsf{F}(\bar{\boldsymbol{x}}) = \bar{\boldsymbol{y}}$ on it but have to ensure *correctness*, i.e., the result $\bar{\boldsymbol{y}} = (\boldsymbol{y}^i, \ldots, \boldsymbol{y}^t)$ has to be shared representation of $\boldsymbol{y} = \mathsf{F}(\boldsymbol{x})$ with $t \geq s$. But for this purpose, we can use according component functions f^i to evaluate F for each share individually. However, finding *correct* component functions is not trivial, in particular if F is a non-linear function [3]. In addition, each component function has to ensure further properties such as *non-completeness* and *uniformity*.

Non-Completeness: As mentioned before, each resulting share $(\boldsymbol{y}^i, \ldots, \boldsymbol{y}^t)$ is given by an individual evaluation of a component function $\mathsf{f}^{i \in \{1,\ldots,t\}}(\cdot)$ over the input shares. However, in order to achieve security in sense of first-order statistical moments, each component function has to provide *non-completeness*. This means that each component function $\mathsf{f}^{i \in \{1,\ldots,t\}}(\cdot)$ must be *non-complete*, i.e., independent of at least one input share.

[3] For a more detailed description, please refer to the original articles [15, 16, 4].

Uniformity: The security of Threshold Implementations as masking schemes is based on the *uniform* distribution of the mask respectively the shared representation which serve as input for a function evaluation. However, since results of a function, e.g., an S-box, are used as input to another function, the outputs of the functions again have to be *uniformly* distributed. This means, given a set of all possible input sharings $\mathcal{X} = \{\bar{\boldsymbol{x}} | \bigoplus_{i=1}^{s} \bar{\boldsymbol{x}} = \boldsymbol{x}\}$ the set of all possible output sharings, i.e., $\{(\mathsf{f}^1(\cdot), \ldots, \mathsf{f}^t(\cdot) | \bar{\boldsymbol{x}} \in \mathcal{X}\}$ should be drawn *uniformly* from the set of $\mathcal{Y} = \{\bar{\boldsymbol{y}} | \bigoplus_{i=1}^{t} \bar{\boldsymbol{y}} = \boldsymbol{y}\}$ as all possible sharings of $\boldsymbol{y} = \mathsf{F}(\boldsymbol{x})$.

2.4 White-Box Cryptography

The concept of *White-Box Cryptography* is concerned with the protection of implementations of cryptographic algorithms in the presence of white-box adversaries that have virtually unlimited capabilities and access to an implementation as well as full control of the execution environment. Implementations are assumes secure against white-box adversaries if they provide an adversary with not more information than given by a black-box access, in other words, the white-box implementation should behave as virtual black box.

As a matter of fact, an ideal white-box implementation of a cryptographic algorithm would consist of a single look-up table which maps every possible plaintext to an according ciphertext (for a given and fixed secret key). However, for modern ciphers that provide security levels and key sizes of 128 bits and more, this approach is obviously impractical. Consequently, alternative approaches which can be realized in practice are necessary. In 2002, Chow et *al.* proposed practical white-box implementations for DES [6] and AES [7] using divide-and-conquer strategies to build white-box implementations using *networks of randomized look-up tables*.

In general, the proposed strategy can be applied for any key-alternating, round-based, symmetric block cipher E_K to derive its white-box implementation E'_K and it can be described as:

$$\mathsf{E}'_K = \underbrace{(\mathsf{f}^{r+1})^{-1} \circ \mathsf{E}^r_{k_r} \circ \mathsf{f}^r}_{table(s)} \circ \cdots \circ \underbrace{(\mathsf{f}^3)^{-1} \circ \mathsf{E}^2_{k_2} \circ \mathsf{f}^2}_{table(s)} \circ \underbrace{(\mathsf{f}^2)^{-1} \circ \mathsf{E}^1_{k_1} \circ \mathsf{f}^1}_{table(s)}$$

$$= (\mathsf{f}^{r+1})^{-1} \circ \mathsf{E}^r_{k_r} \circ \cdots \circ \mathsf{E}^2_{k_2} \circ \mathsf{E}^1_{k_1} \circ \mathsf{f}^1 = (\mathsf{f}^{r+1})^{-1} \circ \mathsf{E}_K \circ \mathsf{f}^1,$$

In this context, $\mathsf{E}^{i \in \{1,\ldots,r\}}$ is a single round of E_K, and $\mathsf{f}^{i \in \{1,\ldots,(r+1)\}}$ are encoding functions which are chosen randomly in order to randomize and hide any key material inside the look-up tables. Besides, in order to ensure full protection of the first and last round of a block cipher and to prevent so called *Code Lifting* attacks [8], white-box implementations usually use external encodings (here given as f^1 respectively $(\mathsf{f}^{r+1})^{-1}$).

3 Methodology

In this section, we introduce our methodology to dynamically update and randomize cryptographic implementations using a generic approach. We first state some important observations that directly lead to a generic representation of the problem. This is followed by an algorithmic solution to achieve dynamic updates of cryptographic implementations.

3.1 Generic Approach

In general, our generic approach can be applied to any cryptographic implementation. However, the provided physical platform has to allow some changes of the implementation during runtime. Since we want to focus on hardware implementations throughout this work, we particularly target reconfigurable hardware in terms of Field-Programmable Gate Arrays (FPGA). Eventually, we present a solution that achieves on-the-fly dynamic randomization of cryptographic implementations.

> **Observation 1.** *Any cryptographic implementation can be represented as network or sequence of modular or atomic functions subsequently applied on an internal state.*

Consequently, we can model any cryptographic implementation as a directed graph. Depending on the level of abstraction and the desired granularity (e.g., system level, gate level, etc.), each node of the graph represents a single or multiple modular and atomic functions of the algorithm. Besides, the edges which connecting the nodes in a certain direction represent the data flow of the internal state.

> **Observation 2.** *Any cryptographic implementation can be modeled by different but equivalent directed graphs.*

In general, the numbers of nodes and edges required to model a cryptographic implementation is not determined and particularly not limited by an upper bound. Principally, we can add new nodes and edges arbitrarily to the graph to find new representations (with sufficient complexity). However, we still have to maintain and ensure correctness of the overall implementation.

3.2 Morphing Algorithm for Cryptographic Implementations

Based on this observations, we developed a generic algorithm to morph a digraph of a cryptographic implementation into an equivalent but encoded digraph while still maintaining correctness of the implementation.

According to Algorithm 1, each arrow $\langle v_i, v_j \rangle$ of a digraph is replaced by an encoded directed edge. For this purpose, both adjacent vertices have to be replaced as well. The starting vertex v_i is replace such that it not only performs

Algorithm 1: Morphing algorithm for cryptographic implementations

Input : $\mathcal{G} = (\mathcal{V}, \mathcal{A})$: digraph representing a cryptographic implementation.
Output: $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{A}^*)$: digraph representing an encoded cryptographic implementation.

$\mathcal{G}^* = (\mathcal{V}^*, \mathcal{A}^*)$: $\mathcal{V}^* \leftarrow \mathcal{V}$, $\mathcal{A}^* \leftarrow \mathcal{A}$

for $\forall v_i \in \mathcal{V}^*$ **do**
  $\mathcal{D} \leftarrow \emptyset$
  $s \leftarrow \mathsf{f}(v_i)$, $\mathcal{V}^* \leftarrow \mathcal{V}^* \setminus \{v_i\}$
  for $\forall v_j \in \mathcal{V}^*$ **do**
    if $a_{ij} \in \mathcal{A}^*$ **then**
      $\mathcal{D} \leftarrow \mathcal{D} \cup \mathsf{f}^{-1}(v_j)$
      $\mathcal{V}^* \leftarrow \mathcal{V}^* \setminus \{v_j\}$, $\mathcal{A}^* \leftarrow \mathcal{A}^* \setminus \{a_{ij}\}$,
    end
  end
  for $\forall d_i \in \mathcal{D}$ **do**
    $\mathcal{V}^* \leftarrow \mathcal{V}^* \cup \{s, d_i\}$, $\mathcal{A}^* \leftarrow \mathcal{A}^* \cup \langle s, d_i \rangle$,
  end
end

return $\mathcal{G}^*$

its originally provided function but in addition performs an encoding function f to the state. In order to maintain correctness of the implementation, the ending vertex v_j has to cancel the applied encoding using the inverse (decoding) function f^{-1} before performing its original function to the state.

3.3 Applicable encoding functions

In this section, we will briefly discuss properties and requirements on encoding functions that are applicable within our algorithm. First of all, the encoding function should be a randomly drawn function in order to perform a randomization of the implementation during the update. However, each encoding function has to fulfill a few minimal requirements and has to provide some properties to be compatible with our methodology. Obviously, the encoding function has to be injective, i.e., it has to be information preserving in order to allow a correct operation of the original implementation. Apart from that, input and output sizes of the encoding functions will depend on the desired granularity of the algorithm and can differ as long as the output size is at least the input size. Besides, the chosen encoding function can have any complexity (but still should be reasonable efficient). Possible realizations of encoding functions could be: linear functions [23], non-linear bijections (like S-boxes) [7], or any other instance which meets the requirements.

3.4 Verification and semantic equivalence checking

Since our methodology should not affect the correctness of the final result of the original implementation, we have to ensure semantic equivalence of the randomized implementations. Therefore, our approach has to include checking and verification steps. As mentioned before, the randomly drawn encoding functions have to meet minimal requirements which has to be checked and verified continuously during the operation. Correctness of the final result, i.e., semantic equivalence of the randomized implementation, is ensured by only encoding single edges (or small paths[4]) and including the inverse decoding function at the same time.

4 Case Study: PRESENT Threshold Implementation

Throughout this section, we present a practical realization of our proposed countermeasure using an encoded PRESENT TI as case study. Before investigating the feasibility of our approach in terms of hiding higher-order side-channel leakage, we give a detailed description of our practical architecture realized on a modern Xilinx FPGA and elaborate our design strategy.

4.1 Adversary Model

Although our practical instantiation employes certain ideas and concepts of White-Box Cryptography in terms of using encoded look-up tables to hide secret key material, we want to emphasize that we still do not consider adversaries of the white-box model. It is obvious that every adversary who has full access and control of the execution environment can circumvent our proposed countermeasures in order to extract secret keys from the implementation using more powerful attacks, e.g., an algebraic analysis of the look-up tables. However, we therefore only consider adversaries of the gray-box model, i.e., adversaries that still can access the implementation but can only gain helpful information through side-channel leakage.

4.2 Design Considerations

PRESENT [5] is a lightweight symmetric block cipher based on a block size of 64 bits. In particular, it is a Substitution-Permutation network (SPN) with 31 rounds. It provides two different key sizes (80 bit or 128 bit) and derives 32 different 64-bit round-keys based on the initial key. Since nowadays, it is advised against using 80-bit keys, we opted to implement and focus on PRESENT-128.

[4] Given for instance a linear operation within a cryptographic implementation (e.g., MixColumns of the AES algorithm) and the application of linear encoding functions would allow to keep encoded intermediate values. However, the decoding function then has to consider the inversion of the linear operation as well.

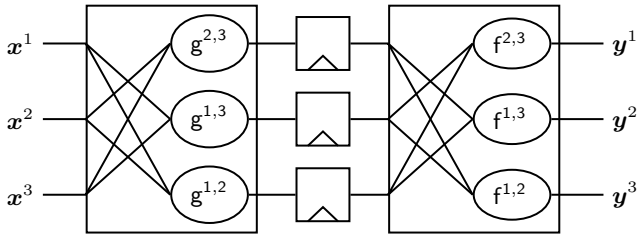


Fig. 1. First-Order TI of the PRESENT S-box

Threshold Implementation of PRESENT: Our implementation is based on the first TI that was presented in [17]. In particular, we apply the decomposition of the S-box into two quadratic functions g and f that was proposed by Poschmann et *al.* in order to benefit from the minimal number of shares (i.e., $m = n = 3$). Since the permutation of the PRESENT cipher is a linear function, it can be applied to each share individually and without modification. Due to the decomposition of the S-box, additional register stages have to be placed in between g and f in order to prevent side-channel leakage caused by glitches. The final structure of the first-order TI of the PRESENT S-box is shown in Figure 1.

Encoding the TI: Before instantiating and implementing our proposed algorithm taking the example of a first-order secure PRESENT TI, we have to find an architecture which supports dynamic updates of sub-functions or components and can be implemented on an FPGA. Given the basic structure of a TI of the PRESENT S-box as shown in Figure 1 we chose the component functions as basic building blocks that have to be updated on-the-fly. Besides, we opted to implement each function as look-up tables because it is a natural choice for FPGAs but also allows fast updates.

Starting from this, the PRESENT TI can be implemented as network of look-up tables, each operating on 4-bit nibbles of the internal state. In a next step, the output of each look-up table is encoded using a non-linear 4-bit bijection. In order to maintain correctness, all subsequent look-up tables have to apply the according decoding function before being evaluated, i.e., the original table has to be combined with the according inverse bijection. In general, this approach reflects basic ideas and concepts of White-Box Cryptography as initially proposed by Chow et *al.* in [6, 7].

However, this strategy has some important implications that effect the final hardware architecture. First, the secret key has to be known during design time since it is included within the look-up tables. Hence, the (shared) key is fixed and combined with the look-up tables of the first layer of the TI S-box. Second, since the permutation layer is a linear functions which operates on single bits, we cannot perform the permutation on 4-bit encoded values. Instead, we have to implement the permutation layer as sequence of look-up tables that decode and re-encode the nibbles while performing the original permutation.

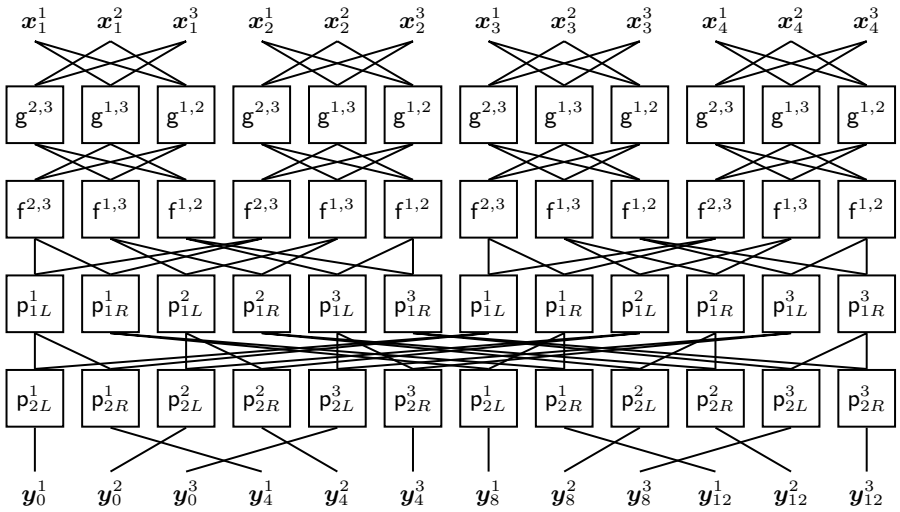


Fig. 2. Quarter Round of Encoded PRESENT TI

Eventually, our encoded TI is implemented using different look-up tables for each sub-function and all rounds. However, this complicates the task of implementing our design efficiently using an round-based approach. None the less, modern FPGAs provide useful features that allow an efficient implementation (as presented in Section 4.3).

Dynamic Update of Encodings So far, our TI is encoded statically using arbitrary non-linear functions applied during design time. However, in order to perform dynamic randomization during operation time, we want to modify these initial encodings. Therefore, in general, we have to find solutions for the following two issues:

1. How to find or compute random non-linear functions on-the-fly?
2. How to inject random non-linear functions into our hardware implementation during runtime?

Random 4-bit non-linear functions, i.e., a random permutations of the sequence $\{0, 1, \ldots, 15\}$ can be generated using a linear-time algorithm using swapping operations and sampling uniform random numbers [22]. Although the permutation generation is slightly biased, this effect can be neglected in the context of side-channel analysis.

Since our encoded TI is implemented as network of look-up tables, injecting random non-linear functions can be realized as table re-computation and re-ordering. In particular, we can apply arbitrary functions to the output of a table by replacing each table entry by the according encoded value. The decoding function can be applied to the input of a table by re-odering the table

Table 1. Area Consumption of our Hardware Architecture

Module/ Component	**Resource Utilization**				
	Logic	*Memory*			*Area*
	(LUT)	*(FF)*	*(LUTRAM)*	*(BRAM)*	*(Slices)*
Control Logic	11	24	0	0	13
Round Function	96	0	0	192	87
g-Layer	0	0	0	48	0
f-Layer	0	0	0	48	0
p_1-Layer	0	0	0	48	0
p_2-Layer	0	0	0	48	0
Reconfiguration	3129	3222	1952	0	2373
Context Engine	22	44	32	0	18
Encoding Engine	2880	2880	1920	0	2258
Randomness Generator	136	256	0	0	40
Total	3236	3246	1952	192	2473

entries according to the decoded address value. Fortunately, this procedure is independent of the previous injection of random functions, i.e., if we first apply a random function n_1 follow by a second function n_2 this is the same as applying another function n_3 with $\mathsf{n}_3 = \mathsf{n}_2 \circ \mathsf{n}_1$. Hence, we can continually update our implementation using random non-linear functions without increasing the size of our implementation by just performing table re-computations and re-orderings.

Eventually, for the given PRESENT implementation, we have to update 5904 4-bit encodings per encryption in order to perform a full *dynamic hardware modification* process. Since there are 16! different 4-bit encodings, the final randomization complexity of our methodology (for the given case study) is about 2^{56}.

4.3 Practical Implementation on Reconfigurable Hardware

The deliberate application of modern reconfigurable hardware in terms of a Xilinx Kintex-7 FPGA provides several interesting advantages and allows a practical evaluation and implementation in order to confirm the feasibility of the proposed approach. In particular, the selected Kintex-7 XC7K160T FPGA implements roughly 12 Mb of block RAM (BRAM) in the form of 325 individual memory instances, each providing 32-Kb of general purpose memory as well as a true dual-port feature. Note that the dual-port option is of particular importance for the dynamic update of our implementation since we can use one port solely to perform the cryptographic operations whereas the second port is used to perform the dynamic table re-ordering and re-computation.

Besides, since all look-up tables of our architecture are 256×4-bit tables, each BRAM primitive could store up to 32 different look-up tables. Fortunately, PRESENT has only 31 different rounds, so we can arrange tables of the same operation but different rounds in the same BRAM instance. This strategy yields a round-based hardware architecture as presented in Figure 2. Moreover, since each BRAM still provides enough memory to store another table we can use this

Table 2. Comparison of different PRESENT Hardware Architectures

Scheme/ Implementation	*Logic* (LUT)	*Memory* (FF)	(LUTRAM)	(BRAM)	*Latency* (cycles)	*Freq.* (MHz)	*Throughput* (MBit/s)	*Ref.*
1st-order TI	808	384	-	-	64	207	413	[14]
2nd-order TI	2245	1680	-	-	128	204	406	[14]
Affine Equivalence	1834	742	-	1	64	112	224	[19]
Glitch-Free Duplication	5442	12672	-	-	704	459	458	[14]
Dynamic Hardware Mod.	3236	3246	1952	192	124	153	315	**new**

free table entry to store an updated table. Hence, after performing the table reordering and re-computation and storing the updated table in the free segment, a context switch is performed, i.e., the storage of the old table is released and the updated table is applied during operation. But since the update is performed through the second port while the first port is continuously used for operation, our strategy does not affect the overall performance.

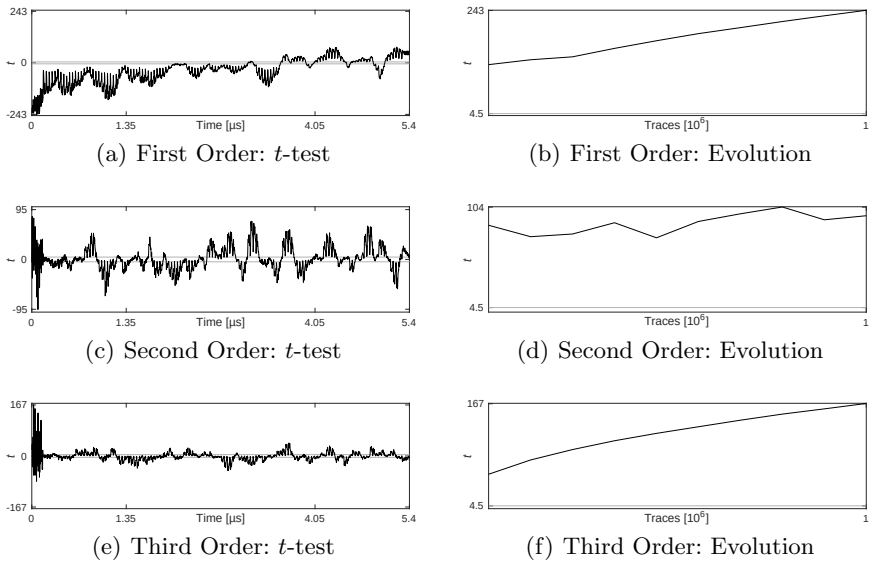
Table 1 provides the implementation numbers of our design, including control logic and a reconfiguration unit that generates new random 4-bit encodings on-the-fly. Obviously, a lion's share of the used resources is necessary to implement the encoding generation. Basically, the round function can be implemented in 192 BRAM instances – the remaining logic in terms of LUTs is necessary to control and operate the table update using the second port of the BRAM. Eventually, the control logic implements a small finite state machine (FSM) that controls both the round function and the reconfiguration engine and provides an interface for external access and control.

4.4 Comparison

In Table 2, we provide a comparison of different approaches to achieve higher-order side-channel resistance (except for the *1st-order TI*) by the example of PRESENT. Obviously, our approach offers competitive results, both in terms of performance and area utilization, although it has an increased demand for BRAM instances. Still, the security of our proposed countermeasure may not only be limited to second-order attacks but it may also affect higher-order leakages, hence providing better security than a *2n-order TI* (at least from a practical point of view).

5 Practical Side-Channel Evaluation

We evaluated side-channel information of our design implemented on a physical device using a SAKURA-X FPGA platform [1] which provides a Xilinx Kintex-7 XC7K160T FPGA for practical side-channel evaluations using the power consumption of the device. Measuring the voltage drop over a 1Ω resistor in the V_{dd} path of the FPGA using a digital oscilloscope with a sampling rate of 500 MS/s, 20 MHz bandwidth limitation, and a stable, jitter-free clock frequency of 24 MHz, we could practically examine vulnerabilities of our proposed design.



(a) First Order: t-test (b) First Order: Evolution

(c) Second Order: t-test (d) Second Order: Evolution

(e) Third Order: t-test (f) Third Order: Evolution

Fig. 3. Non-Specific t-Test Results: Profile 1 (1 000 000 Traces)

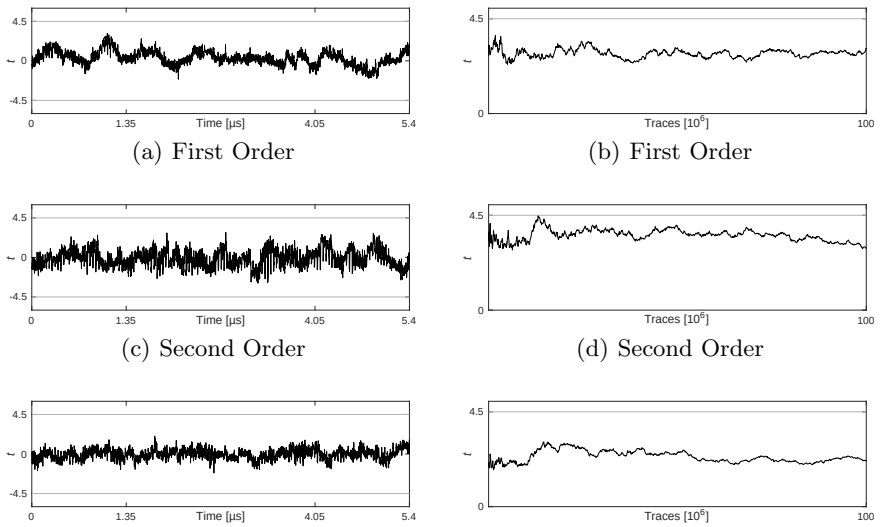
5.1 Non-Specific *t*-Test

A common technique to investigate the resistance and vulnerabilities of physical cryptographic implementations against side-channel attacks is the *Test Vector Leakage Assessment* (TVLA) methodology. The evaluation is based on Welsh's (two-tailed) t-test, sometimes also referred to as *fix vs. random* or *non-specific* t-test, and can be extended naturally to higher-order statistical moments [20, 21].

5.2 Results

In this section we provide practical evaluation results using the non-specific t-test on the first, second and third statistical order. Besides, we include the evolution of the absolute maximum of the t-test over the number of used traces. In total, we performed measurements and evaluations for two different evaluation profiles: first, reference measurements without sharing (i.e., all-zero masks) and omitted dynamic update, and second, measurements using shared values and including our proposed countermeasure in terms of dynamically updating and randomizing the implementation.

Profile 1: Before evaluating the feasibility and effectiveness of our proposed approach, we have to ensure the correctness of our implemented first-order TI using reference measurements. In order to provide such a reference, we measured one million power traces while the PRNG that generates the random masks for



(a) First Order (b) First Order

(c) Second Order (d) Second Order

(e) Third Order (f) Third Order

Fig. 4. Non-Specific t-Test Results: Profile 2 (100 000 000 Traces)

sharing and random encodings was disabled, i.e., all masks were set to zero and the dynamic update was omitted. We expect to detect and observer leakage on all considered statistical orders which is confirmed by our evaluation results shown in Figure 3. One the left-hand side, we provide the results of the non-specific t-test for the first, second and third order after measuring and evaluating the total number of one million traces while on the right-hand side, the development of the absolute maximum for the t-test on each statistical order over the number of evaluated traces is shown.

Profile 2: Eventually, we extend the previous measurement profile by applying our proposed approach in order to hide higher-order side-channel leakages by continuously performing dynamic updates of the look-up tables of our implementation. Again, we do not expect to detect any first-order leakage due to the application of a first-order TI but moreover the leakage detectable at higher statistical orders should be prevented as well. The evaluation results shown in Figure 4 confirm the correctness of these assumptions since we could not detect any leakage after measuring 100 million power traces – neither at the first, second nor third statistical order – which hence also confirms the effectiveness of our proposed approach.

6 Conclusion

In this work we have presented a generic strategy and methodology in order to apply dynamic and random updates to cryptographic implementations and circuits in order to hide higher-order side-channel leakages. Using a case study based on a first-order PRESENT TI and a random updates based on non-linear encodings, we have shown the feasibility and practicability of proposed concept using side-channel power measurements and applying the state-of-the-art leakage assessment methodologies. Eventually, we can conclude that our methodology presents a viable alternative to building higher-order Threshold Implementations and convinces by its generality and scalability.

Acknowledgment

This work has been co-funded by the Commission of the European Communities through the Horizon 2020 program under project number 645622 PQCRYPTO.

References

1. Side-channel AttacK User Reference Architecture. `http://satoh.cs.uec.ac.jp/SAKURA/index.html`.
2. D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM Side-Channel(s). In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, Revised Papers*, pages 29–45, 2002.
3. T. Beyne and B. Bilgin. Uniform First-Order Threshold Implementations. Cryptology ePrint Archive, Report 2016/715, 2016. `http://eprint.iacr.org/2016/715`.
4. B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, and V. Rijmen. Higher-Order Threshold Implementations. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., Proceedings, Part II*, pages 326–343, 2014.
5. A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, Proceedings*, pages 450–466, 2007.
6. S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot. 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, Revised Papers*, pages 1–15, 2002.
7. S. Chow, P. A. Eisen, H. Johnson, and P. C. van Oorschot. White-Box Cryptography and an AES Implementation. In *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, Revised Papers*, pages 250–270, 2002.
8. C. Delerablée, T. Lepoint, P. Paillier, and M. Rivain. White-Box Security Notions for Symmetric Encryption Schemes. In *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, Revised Selected Papers*, pages 247–264, 2013.

9. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, Proceedings*, pages 104–113, 1996.
10. P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, Proceedings*, pages 388–397, 1999.
11. S. Mangard, E. Oswald, and T. Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
12. N. Mentens, B. Gierlichs, and I. Verbauwhede. Power and Fault Analysis Resistance in Hardware through Dynamic Reconfiguration. In *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, Proceedings*, pages 346–362, 2008.
13. A. Moradi and O. Mischke. Comprehensive Evaluation of AES Dual Ciphers as a Side-Channel Countermeasure. In *Information and Communications Security - 15th International Conference, ICICS 2013, Beijing, China, Proceedings*, pages 245–258, 2013.
14. A. Moradi and A. Wild. Assessment of hiding the higher-order leakages in hardware - what are the achievements versus overheads? In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, Proceedings*, pages 453–474, 2015.
15. S. Nikova, C. Rechberger, and V. Rijmen. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, Proceedings*, pages 529–545, 2006.
16. S. Nikova, V. Rijmen, and M. Schläffer. Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches. *J. Cryptology*, 24(2):292–321, 2011.
17. A. Poschmann, A. Moradi, K. Khoo, C. Lim, H. Wang, and S. Ling. Side-Channel Resistant Crypto for Less than 2,300 GE. *J. Cryptology*, 24(2):322–345, 2011.
18. O. Reparaz, B. Bilgin, S. Nikova, B. Gierlichs, and I. Verbauwhede. Consolidating Masking Schemes. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, Proceedings, Part I*, pages 764–783, 2015.
19. P. Sasdrich, A. Moradi, and T. Güneysu. Affine Equivalence and Its Application to Tightening Threshold Implementations. In *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, Revised Selected Papers*, pages 263–276, 2015.
20. T. Schneider and A. Moradi. Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, Proceedings*, pages 495–513, 2015.
21. T. Schneider and A. Moradi. Leakage assessment methodology - Extended version. *J. Cryptographic Engineering*, 6(2):85–99, 2016.
22. N. Veyrat-Charvillon, M. Medwed, S. Kerckhof, and F. Standaert. Shuffling against Side-Channel Attacks: A Comprehensive Study with Cautionary Note. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, Proceedings*, pages 740–757, 2012.
23. Y. Xiao and X. Lai. A Secure Implementation of White-Box AES. In *2nd International Conference on Computer Science and its Applications*, 2009.