Journal of
Low Power Electronics and Applications

MDPI

Article

A Novel Design Flow for a Security-Driven Synthesis of Side-Channel Hardened Cryptographic Modules

Sorin A. Huss [1,2,*] and Oliver Stein [2,3]

[1] Integrated Circuits and Systems Lab, Technische Universität Darmstadt, 64289 Darmstadt, Germany
[2] The Center for Advanced Security Research Darmstadt (CASED), 64293 Darmstadt, Germany; oliver.stein@oth-regensburg.de or oliver.stein@cased.de
[3] Fakultät für Informatik und Mathematik, Ostbayerische Technische Hochschule Regensburg, 93053 Regensburg, Germany
* Correspondence: sorin.huss@cased.de or huss@iss.tu-darmstadt.de

Academic Editors: Osnat Keren, Ilia Polian and Sanu Mathew
Received: 14 May 2016; Accepted: 26 January 2017; Published: 8 February 2017

Abstract: Over the last few decades, computer-aided engineering (CAE) tools have been developed and improved in order to ensure a short time-to-market in the chip design business. Up to now, these design tools do not yet support an integrated design strategy for the development of side-channel-resistant hardware implementations. In order to close this gap, a novel framework named AMASIVE (Adaptable Modular Autonomous SIde-Channel Vulnerability Evaluator) was developed. It supports the designer in implementing devices hardened against power attacks by exploiting novel security-driven synthesis methods. The article at hand can be seen as the second of the two contributions that address the AMASIVE framework. While the first one describes how the framework automatically detects vulnerabilities against power attacks, the second one explains how a design can be hardened in an automatic way by means of appropriate countermeasures, which are tailored to the identified weaknesses. In addition to the theoretical introduction of the fundamental concepts, we demonstrate an application to the hardening of a complete hardware implementation of the block cipher PRESENT.

Keywords: side-channel analysis; secure CAE design

1. Introduction

Nowadays, embedded devices find application in an increasing part of everyday life. Due to the on-going progress of the fabrication technology, integrated devices get more and more powerful in terms of computational throughput, as well as in terms of complexity. However, the increasing number of embedded devices also gives rise to potential unintended exploitation. In order to secure the devices against misuse and create a secure communication environment, various protocols and cryptographic schemes have been developed in the meantime.

In the last two decades, a new attack type emerged, targeting directly implementations of devices that process sensitive data. This kind of assault is called the side-channel analysis attack, which exploits physical observables that depend on the processed data. Side-channel attacks are especially dangerous due to their general applicability to cryptographic schemes and their non-invasive nature. Thus, a side-channel attack can be performed on various implementations, and a compromised secret is very difficult to detect. Examples for devices targeted are smart cards, USB security tokens, RFID tags or even complex cryptographic co-processors. We focus in this contribution on the design of hardened hardware partitions within an embedded system, since security applications often require considerable computing power, which in many cases is best available from dedicated hardware modules.

Various countermeasures have been suggested in order to harden embedded devices against such attacks. These countermeasures are often specifically adapted to an algorithm, implementation or device, thus limiting their general applicability. Furthermore, it is nearly impossible for a designer to predict a-priori the side-channel resistance of an embedded device before its detailed implementation and a subsequent in-depth analysis. Thus, countermeasures against side-channel attacks are often developed and implemented in an additional step at the end of the design process of a device. At this stage, however, the available resources on the device are rather limited, forcing the designer to either consider less efficient countermeasures or to perform a major redesign of the device. Figure 1 depicts the usual design flow of an embedded system commonly used in a security-sensitive application. Usually, the design of a side-channel-resistant hardware module is done in two separate phases, which may lead to a cyclic process. First, the module's functionality is specified in a hardware description language (HDL), for instance VHDL, and its correctness is tested by functional simulation. After passing the functional tests, which only ensure an error-free behavior in the normal operation mode within the specification parameters of the device, the implementation phase starts. This phase makes use of CAE tools that assist the designer in translating a high-level specification into a more hardware-related description. If the designer has an appropriate model of the target technology platform, a security engineer can then perform a first side-channel evaluation of the hardware design. Usually, this side-channel analysis is done after the implementation phase of the design and is based on a prototype for the physical verification test.

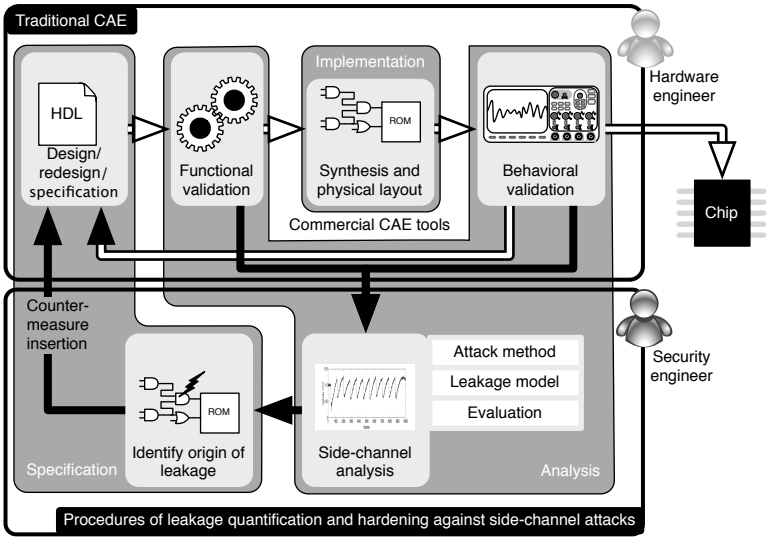

Figure 1. Traditional side-channel-aware hardware design flow and engineering roles.

During the side-channel evaluation, the hardware designer provides all necessary information to the security engineer. Then, the security analyst has to select an appropriate attack method that depends on the threat scenario of the circuit's application. Additionally, he/she has to construct a physical model of the device that is as precise as possible and describes the expected exploitable information leakage. Both the model and the prototype are then used in the evaluation phase. This phase aims at identifying the observable side-channel leakage and, if possible, also its origin. If a successful side-channel attack is identified, then this information can be used to restart the design cycle. If, on the other hand, the exploitable side-channel leakage is too low to reveal the circuits' secret or the effort in order to recover this secret is too high for the targeting security-sensitive-application, the target device is deemed side-channel resistant. Such a design flow requires at least a designer, who is able to take both engineering roles as detailed above. This person has consequently to be skilled both in side-channel analysis and hardware design methodologies. However, human designers who have a sufficient expertise in both areas are hard to find.

In this paper, we mainly investigate the question of how to add both an automated and supportive side-channel evaluation and a dedicated synthesis method to the conventional CAE tool set. We thereby propose a framework that supports both, the security, as well as the hardware engineering role in the construction of side-channel hardened devices. On the one hand, the framework is able to automatically analyze a given HDL coded design specification in terms of information and data flow and, on the other hand, to automatically embed first-order countermeasures selected by a human designer into the original HDL code. By manipulating the HDL description of the circuit, this method can be applied to various implementation platforms ranging from FPGAs to ASICs.

The paper is structured as follows. Section 2 provides some background information about side-channel analysis in terms of possible attacks and well-known countermeasures, as well as an overview of related work. Section 3 introduces the architecture of the AMASIVE (Adaptable Modular Autonomous SIde-Channel Vulnerability Evaluator) framework, which supports a designer in the process to implement side-channel hardened devices. Section 4 then introduces the reshaping supportive method, which may be viewed as a major step towards a new methodology that automatically embeds appropriate countermeasures in an initial functional design. In Section 5, we demonstrate the advantages of the proposed approach with respect to the analysis and the subsequent hardening of a PRESENT block cipher implementation on top of a Xilinx Virtex 5 FPGA platform. Finally, Section 6 concludes the paper.

2. Background on Side-Channel Analysis and Related Work

In this section, we briefly explain the ideas behind power analysis attacks and the most important countermeasures for the reader who is not familiar with this kind of side-channel attack. Moreover, we give an overview of existing approaches with a similar purpose.

2.1. Power Analysis Attacks

One of the most common side-channel analysis attacks nowadays is the power analysis attacks. Power analysis attacks exploit the data-dependent switching activity of a cryptographic implementation in order to extract a secret key of intermediate values of internal operations or of the current internal state of the algorithm. The most common power analysis attack, the Correlation Power Analysis (CPA) attack [1], is currently available in AMASIVE. It aims to reveal the subkey from the captured traces by using the Pearson correlation as the statistical tool. We refer to [2] for an overview of side-channel analysis attacks and [3] for a detailed introduction to such attacks.

2.2. Countermeasures

Since the introduction of power analysis attacks, various countermeasures have been studied and developed. An intuitive approach for a countermeasure is to decrease the exploitable power consumption of the device. There are mainly two different basic approaches that may reduce the exploitable information leakage due to the data-dependent power consumption. The first approach, hiding, focuses on the physical behavior of the implementation. The principle of hiding is to decouple the data-specific power consumption from the actual internal processed intermediate values.

The second approach, masking, reduces the degree of the exploitable data-dependent power consumption by internally randomizing the processed intermediate values by using auxiliary or masked values instead. In order to compute the correct output, the intermediate results have to be corrected at the end of the cryptographic algorithm run. Without knowing the mask value, the adversary cannot construct correct hypotheses for the data-dependent power consumption, thereby making a side-channel attack much more complex. An overview of some important countermeasure techniques is provided in Table 1. We refer to [3] for more information on countermeasures against power analysis attacks.

Table 1. Classification of important countermeasures.

Countermeasure	Class		Level	
	Hiding	Masking	Algorithm	Cell
Shuffling [3,4]	x		x	
Dummy operations [5]	x		x	
Dual-rail-logic [6,7]	x			x
Register pre-charging [8,9]	x		x	
Random clock [10]	x		(x)	x
TImasking [11,12]		x	x	
Boolean masking [13]		x	x	
USMmasking [14]		x	x	
MDPL [15,16]	x	x		x

2.3. Hardening against Side-Channel Attacks

In contemporary hardware designs, the resistance of the device to side-channel attacks is often considered at the end of the design process only, which makes this important step more dependent on available resources than on actual security requirements, cf. [17]. Such a late consideration is especially disadvantageous as it forces the designer to rerun the overall design process in order to include side-channel countermeasures. The reasons for a rather late consideration of side-channel vulnerabilities are manifold: short time to market, unpredictability of side-channel properties during the standard design process or an unidentified side-channel information leakage of the underlying algorithm. Thus, side-channel vulnerabilities are detected too late to be taken into account when deciding on the design architecture.

However, taking such flaws too late into consideration is rather unfortunate, because the designer has access to an enormous amount of knowledge about the device under construction. Therefore, she or he is in the position to perform a much more accurate side-channel analysis than an outside attacker ever could. However, identifying, combining, and utilizing this knowledge is a difficult task and requires experience in side-channel analysis next to hardware design skills.

Recently, there have been some contributions from academia that aim at a timely support of the designer in securing a device against side-channel attacks in both software and hardware implementations. The securing support can be classified into two different fields of research investigations. One research field deals with improving the analysis method in order to detect exploitable side-channel leaks of the implementation under analysis. Hence, this more investigated research branch focuses on statistical analysis methods and their comparison, cf. [18–21]. The other research field is concerned with the support of embedding countermeasures in order to strengthen the implementation's resistance against side-channel exploitation. Countermeasures against side-channel attacks in software implementations of crypto modules are in the focus of three approaches, which provide automated code analysis and countermeasure integration, cf. [17,22–24]. The work in [17] applies a side-channel attack to the assembled code and utilizes an estimation of the mutual information [18] to evaluate and to highlight vulnerabilities of a specific code section. The second method, proposed by Moss et al. in [22,23], features an automatic insertion of masks by evaluating the secrecy requirements of an intermediate value in the program code.

The first step towards an intelligent insertion of side-channel-related countermeasures during the design phase of a hardware implementation was proposed in [24]. Here, Regazzoni et al. introduced a tool set that automatically transforms each element of the design net list into the logic style MCML (MOS Current Mode Logic) deemed to be more resistant to side-channel power attacks than common logic styles. The exchange of vulnerable net-list elements is based on an evaluation aimed at identifying security-sensitive parts, which need to be hardened by means of this specific logic style. Recently, Bayrak et al. extended the framework introduced in [17], see [25]. In addition

to random precharching, they introduced Boolean masking as an advanced countermeasure, which assumes a global code transformation.

In two of these three methods, the designer has to formulate the leakage assumption for the side-channel vulnerabilities, which presumes a designer with considerable expertise in the attack field. In fact, the designer has to anticipate which attacks a potential adversary will conduct on her or his design. The proposal in [17] overcomes this problem by utilizing an estimation of the mutual information as a side-channel distinguisher.

Since the selected leakage model has the greatest impact on the success of the attack, cf. [19,26], it seems to be reasonable to select the best model for each attack scenario. In addition, the leakage model relies on an adequate hypothesis function definition in order to identify possible vulnerabilities in the design. In other words, approaches that support these desirable features are more flexible and modular with respect to side-channel analysis tools and of course in terms of various automatically-embedded countermeasures than the previously-proposed schemes. Due to an modularized library approach, more countermeasure schemes can be added to the framework. The proposed framework AMASIVE aims to remove some of the disadvantages of the existing methods stated above and to take the previously mentioned general design criteria into account.

3. The AMASIVE Framework

The advocated design framework is a means to support the designer in a comprehensive way when developing side-channel resistant cryptographic devices. Similar to [18], it is based on an an attacker model for the security analysis. AMASIVE requires the designer to specify the attacker model by stating:

- public values (input/output) of the algorithm,
- security-sensitive values (keys),
- the model function, which describes the power consumption of the device,
- computational boundaries of the attacker and
- attacks, which an adversary is able to use.

Compared to common design flows of hardware designs and the above-mentioned frameworks, AMASIVE offers the following new features:

- It can be used to automatically identify side-channel vulnerabilities at different stages of the design flow of embedded systems.
- It suggests various countermeasures early in the hardware design process.
- It integrates previously chosen countermeasures autonomously into the design, which in turn can be analyzed again by the framework.
- It assists a hardware designer without deep knowledge in power analysis attacks to perform both a security analysis and a secure redesign, within the actual design phase.
- It provides great flexibility since it can be adapted to various implementation platforms, leakage models and side-channel distinguishers. Please note that this version of AMASIVE focuses on block cipher modules; an extension to other classes of cryptographic algorithms is envisaged in our future work.

Therefore, to the best of our knowledge, the AMASIVE framework is the first approach to automatically identify vulnerabilities against power analysis attacks, to suggest and to autonomously add countermeasures to a design at an early stage of the design activities.

Note that the changes made by AMASIVE are fully under designer control. They depend both on the design at hand and on the countermeasures chosen by the designer. After a subsequent security analysis, which evaluates the effectiveness of the introduced countermeasures, the designer decides whether the envisaged security requirements are accomplished by the inserted countermeasures.

Thus, the designer is controlling a much more flexible design process, which enables her or him to directly adapt the design to given security requirements.

3.1. Autonomous Side-Channel Analysis

The basic idea of the AMASIVE framework relies on a multistage approach of security analysis and countermeasure suggestions (cf. Figure 2). Both the security analysis and countermeasure proposals are intended to interact with each other. The related security analysis is presented and discussed in [27]. For the convenience of the reader, we describe in some detail the security analysis performed within the AMASIVE framework.

The analysis process can be subdivided into the information collection, the graph representation, and the vulnerability analysis phase, respectively:

1. We utilize various information sources in order to collect all necessary information for the later analysis. These sources depend on the envisaged implementation platform of the hardware design. So far, designs denoted in VHDL can be handled. However, AMASIVE can easily be adapted to work on designs modeled in a different HDL, such as Verilog or even SystemC on the RT level.
2. The required information is collected from the VHDL source code, simulation results and the designer. It is mainly used to construct a graph G. This graph captures both the data flow and the hardware architecture of a given VHDL model.
3. An attacker model is defined, which specifies the assumed capabilities of the attacker (see the beginning of Section 3), and a complete analysis is then performed based on both the constructed graph and on the selected attacker model.

In total, we distinguish five basic graph elements:

1. Register: The register element models the storage of an intermediate value. A register element is described by the number of bits it can store at one point in time and an ID in order to identify register transitions.
2. Operations: The operation element models a generic module that modifies data. Each operation is linked to a functional description represented either by a look-up table or by a C code description. Additional properties, e.g., the number of input/output bits or information about whether the processed function is invertible and bilinear, etc., are included. Thus, the operations model the functional representation of the implementation.
3. Permutation: A permutation is considered as a separate element since, e.g., on an FPGA, it is performed by a simple reordering of wires. Similar to an operation, a permutation element is described via a two-dimensional look-up table and a bit size value.
4. Entropy: The entropy element represents secret information that is inserted during the execution of the algorithm, such as the key or a random number. During the execution of the encryption algorithm, entropy elements are usually the information that has to be protected against an adversary. The adversary, on the other hand, aims at recovering these elements. For each entropy element, we store the bit size, as well as the respective entropy, which states the uncertainty of an adversary regarding the actual value.
5. Channel: Channel elements connect the above-stated graph elements. Thus, the channels model the structural information of the hardware architecture. In particular, they represent the physical connections between the single modules of the implementation platform.

The graph G is then generated from the nodes and the edges given by the channel elements. Therefore, on the one hand, G models the hardware structure of the design via the nodes representing entities and components of the underlying VHDL code and via the edges representing the signals that connect the entities and components. On the other hand, G also models the information flow of the design since each node denoting an operation is linked to a functional description of the corresponding processing module.

For the vulnerability analysis, the designer has to determine the public values (input/output), the security-sensitive values, and the computational boundaries of the attacker. Currently, the

framework provides the following actions an attacker can perform to recover the key or the secret intermediate values:

1. Deduction: Given an operation op and an input/output pair (i, o) with $o = op(i)$, perform:

$$\begin{cases} op(i), & \text{if } i \text{ is known,} \\ op^{-1}(o), & \text{if } o \text{ is known and } op \text{ invertible.} \end{cases}$$

   If the operation op has no input, then a deduction just performs the operation.
2. Reduction: Reduce the entropy of a given node if this node is connected to another node with reduced entropy (for instance, if i is unknown, parts of e are known and o is known, then the equation $i \oplus e = o$ allows the reduction of the entropy of i).
3. Power attack based on a hypothesis function: The framework provides automatically a leakage function $\ell : \{0,1\}^b \to \mathbb{R}$, which is based on an appropriate model for the power consumption. Currently, the Hamming Weight (HW) model and Hamming Distance (HD) model are implemented. Typically, the proposed hypothesis function ℓ is stated in terms of the model chosen by the designer and appropriate operations with known input or output, which also process a secret value. These operations are obtained by the above described analysis of the graph (for instance, a very common operation for a block cipher is $S(x \oplus k)$ or $S^{-1}(x \oplus k)$, where is S is an S-box, x an input and k a part of the key).

Algorithm 2 in [27] explains how a hypothesis function in terms of the Hamming weight model is identified. By checking every step of Algorithm 2, it turns out that the model function $HW : \{0,1\}^b \longrightarrow \mathbb{R}$ can be replaced by any suitable model function $m : \{0,1\}^b \longrightarrow \mathbb{R}$.

Similarly, it can be verified in the same way that Algorithm 3 in [27], which yields a leakage function ℓ based on the Hamming distance model, works also for any other model function $M : \{0,1\}^b \longrightarrow \mathbb{R}$.

It is also possible to consider complex model functions $m : \{0,1\}^b \longrightarrow \mathbb{R}$, $m = \sum_{l=1}^{u} \beta_l \cdot m_l$, where $\{m_l : \{0,1\}^b \longrightarrow \mathbb{R}, \quad l = 1, \ldots, u\}$ is a family of model functions. Therefore, it is possible to verify the vulnerability or the resistance of a hardware design against power attacks based on higher dimensional leakage functions. Again, Algorithm 2 or 3 can be applied to check whether m is a candidate for a hypothesis function.

As pointed out in [27], after the automatic generation of a hypothesis function, a power attack (CPA in this context) is performed to determine the practical feasibility of the theoretical attack based on the graph analysis. In particular, the feasibility of each hypothesis function is verified by checking whether the required number of measurements is within a given complexity boundary (which is provided by the designer and is also the basis for the decisions in Algorithms 2 and 3). Clearly, a different distinguisher or even different types of power attacks like stochastic methods or template attacks based on a hypothesis function can be selected, provided they are already available in AMASIVE.

An example of the just sketched security analysis is highlighted in [27]. The example is based on the lightweight block cipher PRESENT, which is an interesting candidate for embedded systems. However, as PRESENT shares the basic building blocks (i.e., S-boxes, permutations and XORs) with most block ciphers, so this discussion is quite general in the context of block ciphers.

The output of the security analysis, i.e., the hypothesis function and the involved components of the architecture, is first presented to the designer and then forwarded to the countermeasure suggestion module.

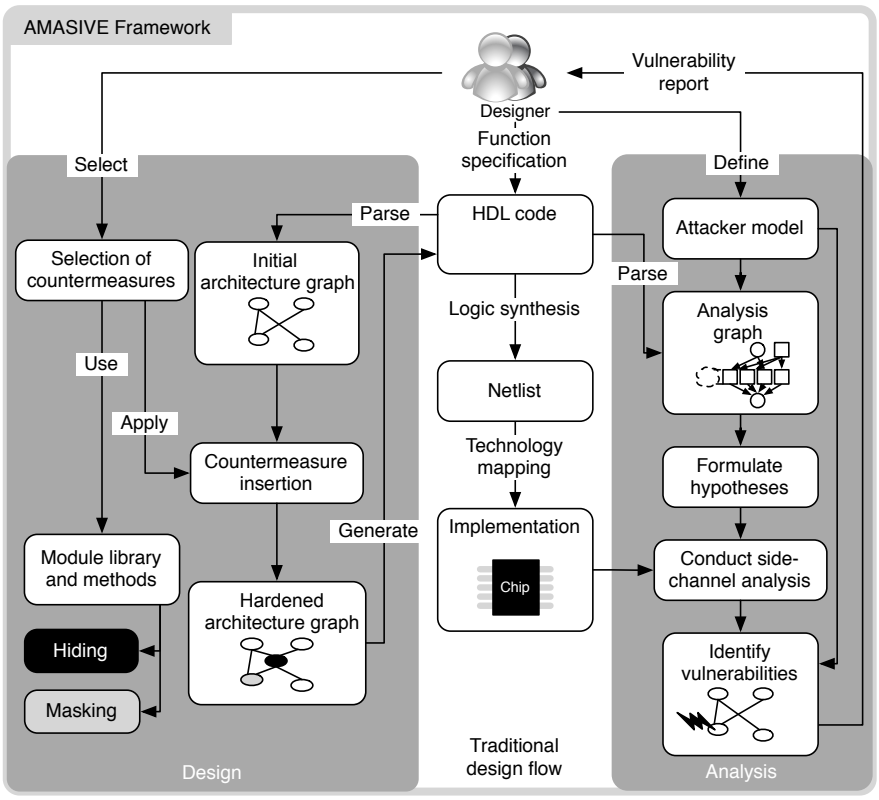

Figure 2. Overview of the AMASIVE (Adaptable Modular Autonomous SIde-Channel Vulnerability Evaluator) workflow.

3.2. Automatic Embedding of Countermeasures

In this subsection, we give an overview of the steps that are to be performed within the AMASIVE framework in order to add countermeasures to a hardware design.

The security analysis provides the components that are susceptible to a power analysis attack chosen by the designer. The designer has the possibility to select from the available countermeasures and those vulnerable components to which the chosen countermeasures should be applied. The selected countermeasure(s) and components are subsequently given to the countermeasure module. The available countermeasures are stored in a data base and can be selected interactively by the designer. Currently, the related countermeasures are stored as entities and components, respectively.

The insertion of a countermeasure requires a major change in the logical structure of a design. The approach of AMASIVE is to perform these changes not on the level of the HDL source itself, but on an abstract model of the structure of the source code. This model is realized by means of Python objects. A model in terms of Python objects is well-suited in this context, since Python provides both convenient tools to generate and manipulate objects and powerful tools to collect information from an HDL source file. The hardened model is subsequently translated back to an HDL file.

Consequently, in a first step, the HDL code is parsed to extract the architectural description of the design. In particular, every entity of the data path is captured. Along with each entity, all components, ports and signals are collected. Moreover, the connections between all of these elements are identified and recorded.

For each of these HDL elements, a Python object is created, which contains a name, the "type" of the corresponding element and all logical connections to other structural elements. These objects and their connections form a graph $\hat{G}$, which from now on is referred to as the architecture graph. Thus, each object represents a node and the connections between objects form the edges of $\hat{G}$.

The insertion of new and the replacement of present components within a given architecture coded in HDL comprises the key operations for the implementation of countermeasures against power attacks.

The insertion of a new component results in adding the corresponding object to the graph $\hat{G}$. To this end, this object has to be generated, and the port declaration of those objects, between which the new object should be placed, have to be adjusted accordingly.

Replacing a component by another one is a similar process. The component to be replaced has to be specified and the complete replacing component to be either generated or taken from the data base. A detailed description of these operations can be found in Section 4.

After all changes aimed at hardening the design are completed, the HDL source file reflecting these changes is generated from the modified graph as depicted in the left-hand part of Figure 3. The whole process is summarized in Algorithm 1.

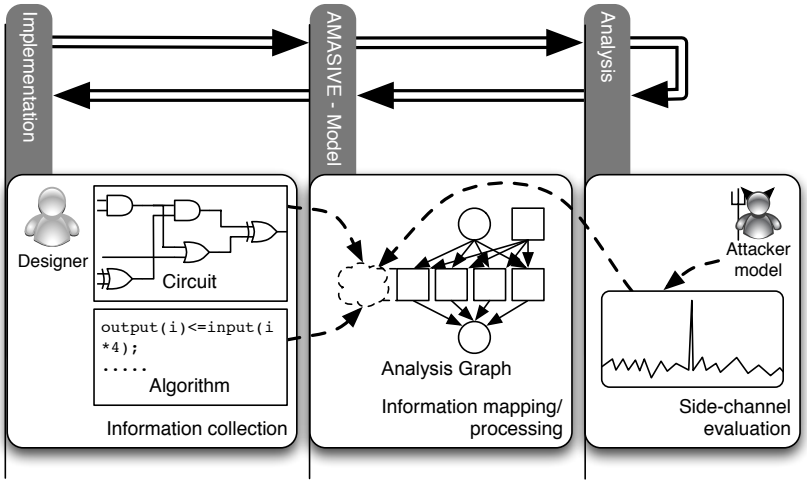

Figure 3. Architecture of the AMASIVE framework, cf. [27].

Algorithm 1 Generate a side-channel hardened circuit.

Require: VHDL code attributed by AMASIVE identifiers and the inputs of the security analysis

1: Parse the given design code hierarchically beginning with the data path module, which is marked with the AMASIVE identifier *data path*
2: Generate $\hat{G}(V, E)$ based on VHDL code
3: Enhancement of $\hat{G}(V, E)$:
4: Select a node $V_j \in V$, which should be manipulated by embedding a side-channel countermeasure
5: Manipulate $\hat{G}_{Sec}(V, E) \leftarrow \hat{G}(V, E)$ by establishing new edges E_{+i} to embedded necessary nodes V_{+i} in $\hat{G}(V, E)$
6: Generate new VHDL code (VHDL$_{Sec}$), based on the original VHDL code, secured VHDL macros, and the new $\hat{G}_{Sec}(V, E)$
7: (Re-)Synthesis of the side-channel attack hardened circuit based on new VHDL$_{Sec}$ code.

4. Security-Driven (Re-)Synthesis

This section aims to detail the description of the autonomous integration of the countermeasures of the AMASIVE framework.

In the first subsection, we discuss in depth the architecture graph $\hat{G}$ of the cryptographic module at hand and detail the process of inserting several new components in a structured way into the design code. This process is based on the insertion routine, which was already mentioned in the last section and which is by far the most frequently-used operation for the integration of countermeasures. In the next subsection, we present the so far implemented countermeasures and indicate how they are

composed of the beforehand described routines. Additionally, we address how these countermeasures can be used within the AMASIVE framework, cf. Section 4.2.1.

4.1. Architecture Graph and Countermeasures

As outlined before, the basis of the countermeasure insertion of the AMASIVE framework is the graph $\hat{G}$.

Compared to the analysis graph G as introduced in Section 3, the architecture graph can be interpreted as a subgraph of G. While $\hat{G}$ represents the structural information of a hardware design based on the VHDL source code only, the graph G also models the information and data flow within the algorithm based on additional data sources. Each node V_i of $\hat{G}(V, E)$ represents an element in the VHDL code, in particular entities, contributing to the hardware structure of the design. Each edge E_i of $\hat{G}$ denotes a signal interconnection between these entities.

Being more precise, the architecture graph features five different types of nodes that represent elementary components of circuits, which are used in a cryptographic algorithm. Table 2 lists all nodes and the related components, as well as additional entries. Currently, parameter flags in the VHDL code, the so-called identifiers, are used to provide the necessary information of to which type of node the entity should be assigned. Other additional information, provided by the data path identifier, is the top-level module in the VHDL code, which contains the complete data path. By assigning the AMASIVE identifier to the investigating data path, the framework can autonomously generate the architecture graph of the core part of the system.

Table 2. Types of nodes in the AMASIVE architecture graph.

	Node Type				
	Register	**Switch**	**Permutation**	**Non-Linear Function**	**Data Path**
Component	Flip flop	Multiplexer	Permutation module	Substitution module	Top level module
Identifier	*register (clk, rst, en)*	*mux (sel)*	*permutation*	*non-linear*	*data path*
Optional identifier attributes	clk, rst, en	sel	-	bilinear, invertible	-

There are several functions available in the framework for the automated exchange of the data path. Two essential operations are used to insert nodes into $\hat{G}(V, E)$ next to or in between existing nodes. A combination of these insertions allows one to add more complex structures to the existing $\hat{G}(V, E)$ and thereby manipulates the original architecture graph to a large extent. The combined application in form of different orders and numbers of iterations can be applied to generate more complex structures, such as mirroring a complete data path and thus allowing Boolean masking. Algorithms 2 and 3 denote these two basic functions, which are necessary to embed the various countermeasures. Both algorithms handle the placement and the routing of the entities of the original design, as well as of the additional components of the countermeasure circuit.

Algorithm 2 Add_Seq: Add new k nodes $V_{+s,i}$, $i = 1, \ldots, k$ sequentially between existing nodes V_{pre} and V_{succ}.

Require: Architecture graph $\hat{G} = (V, E)$ and node V_j that becomes the successor V_{succ} after $V_{+s,i}$ are placed
1: $V_{succ} \leftarrow V_j$ and V_{pre} are all preceding nodes V, which are connected to V_j
2: Collect all E_i between V_{succ} and V_{pre}
3: Reroute E_i to connect V_{pre} and $V_{+s,i}$
4: Create new edges $E_{+,i}$ to connect $V_{+s,i}$ and V_{succ}
5: **return** $\hat{G}_+ = (V, E) = (\{V, V_{+s,i}\}, E)$

Algorithm 3 Add_Par: Add new k nodes $V_{+p,i}$, $i = 1, \ldots, k$ in parallel to an existing node V_j.

Require: Architecture graph $\hat{G} = (V, E)$ and node V_j working in parallel to node $V_{+p,i}$
1: Identify successor node V_{succ} and predecessor node V_{succ} of V_j
2: Add a component to merge two edges into a single one between V_j and V_{succ} by using Algorithm 2
3: Create a new edge E_+ to add the new nodes $V_{+p,i}$ to the graph with predecessor nodes V_{pre} and V_{succ}
4: **return** $\hat{G}_+ = (V, E) = (\{V, V_{+p,i}\}, E)$

4.2. Embedding Selected Countermeasures

These basic manipulation functions of the architecture are essential to embed various countermeasures. Please note that the outlined algorithms are quite similar to linked list processing methods, thus featuring similar properties with respect to, e.g., scalability or storage space requirements.

Currently, the AMASIVE data base supports the following fundamental countermeasures. More details on such countermeasures may be found in the references summarized in Table 1.

- Random register switching
- Component masking
- Boolean masking of data paths

By means of Algorithms 2 and 3, these generic countermeasures are automatically inserted into the original design, whereas the current version of the tool set does not yet support advanced features, such as interleaved exchange of masks. For instance, Algorithm 2 provides the sequential placement of XOR operations in order to provide Boolean component masking to the circuit. After the application of Algorithm 2 to the two XOR components, Algorithm 3 is used to place a copy of the masked function in parallel to the original one for calculating the correction term, cf. the center illustration of Figure 4. More complex countermeasures can easily be generated by using appropriate combinations of these algorithms and additional predesigned side-channel hardened components stored in the library of AMASIVE.

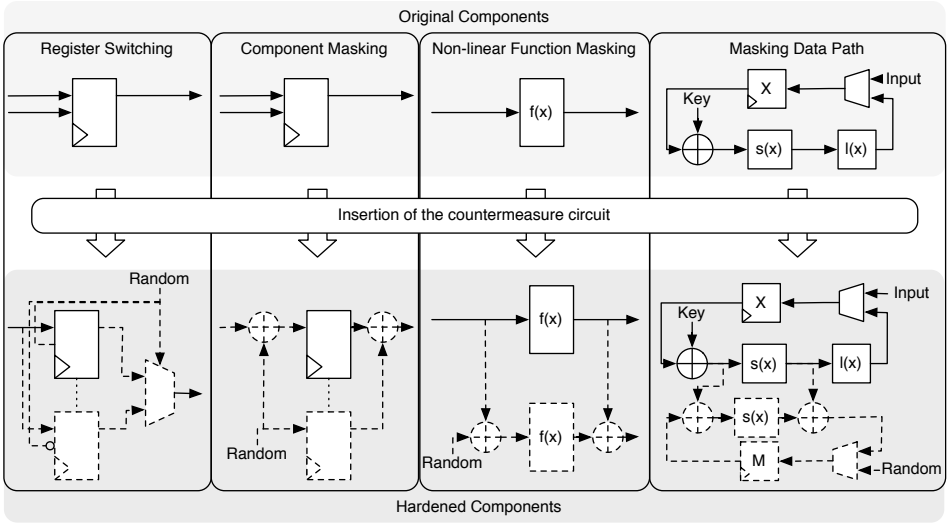

Figure 4. Component transformation within security-driven synthesis.

4.2.1. Granularity of Inserted Countermeasures

As already pointed out, all of these countermeasures can be embedded fully automatically in the original data path without manually writing any VHDL code segment. For the embedding of the

countermeasure, the designer only has to write the name of the initialization label of the component he or she wants to harden (typically those components that were identified by the security analysis) next to the side-channel countermeasure to be applied into the same line of the configuration script. The countermeasures can be inserted into the original circuit description at considerably different levels of granularity. For instance, he or she can apply the random switching register countermeasure to all state registers of a block cipher or individually for selected registers only. Thereby, the designer is able to vary the trade-off between resource consumption and security level denoted by the minimum number of traces required to unveil the whole secret. The transformation of original components to hardened versions as supported in the actual framework version is visualized in Figure 4.

4.2.2. Plugin Structure for Future Extensions

As already mentioned in the Introduction, it is envisaged to extend AMASIVE in various ways. In this section, we point out how the advocated framework supports such extensions.

The concept of a library of hardened components goes hand in hand with the basic idea to keep this framework modularized. For each hardening method, just two implementation steps are necessary to add a new countermeasure to the framework. The first step is to represent the embedding process of the countermeasure in a script file. The other step is to make the required hardened components available, which are then inserted or replaced by the script provided in the first step. Due to this separation, existing countermeasures can easily be extended. Therefore, the masked S-box construction on the right-hand side of Figure 4 may simply be replaced by another table look-up masked S-box version denoted $S'(x)$.

Each of these countermeasures needs a source of entropy, which may be embodied either by an external source or by an additional component description coded in VHDL. When using an external source, an additional port in the top-level module will be generated. In the other case, an LFSRis being added to the design and then properly connected. The hardened modules are subsequently saved as synthesizable VHDL files in the countermeasure library, thus allowing for a very flexible application to different implementation platforms, such as various FPGAs or ASICs.

5. Evaluation Results for the PRESENT Block Cipher

In this section, we demonstrate the hardening of the block cipher PRESENT [28] by exercising the outlined approach.

We decided to address PRESENT instead of the well-known AES algorithm mainly because "[...] the AES is not suitable for extremely constrained environments such as RFID tags and sensor networks [...]" as Bogdanov et al. pointed out in [28], which is a major application area for secure embedded systems design. In our opinion, the usual trade-off between the resources dedicated to the integrated countermeasures and the achieved security level becomes more visible by means of an ultra-lightweight block cipher demonstrator.

For the sake of completeness, we briefly recapitulate the preceding security analysis of PRESENT. The detailed analysis and information on the utilized PRESENT version can be found in [27]. More implementation details may also be found in the related research project documentation [29].

5.1. Analysis of the Unprotected PRESENT Version

As noted in Section 3.1 and in [27], it is necessary for the security analysis that the designer provides some information, in particular the secret and known values, i.e., entropy and register elements and the capabilities of the attacker. In our case study, the designer assumes that the attacker has knowledge of both the plain and the cipher text, which is a common scenario in side-channel analysis. It is also presumed that the attacker is able to run a brute-force attack up to a complexity of 2^{32} and to execute a deduction and a reduction operation, respectively. Finally, the designer grants the attacker the ability to perform a CPA using the Hamming distance HD as a model function, which is well suited for block cipher implementations on an FPGA, i.e., it represents the only power

attack currently available in AMASIVE. Subsequently, all known and secret intermediate values of the algorithm have to be calculated, and to each of these, the corresponding entropy value has to be assigned.

Figure 5 details the processing of four four-bit blocks of both the first and the last round of PRESENT. A fragment of the corresponding analysis graph is visualized in Figure 5c). The last round is the starting point of the security analyis. We selected the Hamming distance model for the power analysis. Therefore, Algorithm 3 in [27] can be utilized to construct and specify a leakage function. Since the framework starts the analysis in the last round, the first unknown intermediate values are stored in the register r^{30}, and the leakage function ℓ is identified as:

$$\ell(r^{30}, r^{31}) = HD(r_0^{30}, r_0^{31}) || \cdots || HD(r_{15}^{30}, r_{15}^{31}) \tag{1}$$

with:

$$r_0^{30} = S^{-1}(s_{0,0}^{31} || s_{4,0}^{31} || s_{8,0}^{31} || s_{12,0}^{31})$$

and accordingly for $r_i^{30},\ i = 1, \ldots, 15$, cf. Figure 5b).

Here, r_i^{30}, r_i^{31} denote the i-th four-bit block of the registers r^{30} and r^{31}, and $s_{j,b}^{31} = o_{j,b} \oplus e_{j,b}^{31}$, where $o_{j,b}$ is an output node and $e_{j,b}$ denotes an entropy node representing the last round key. AMASIVE determines this leakage function together with the affected register entities and outputs both as a string.

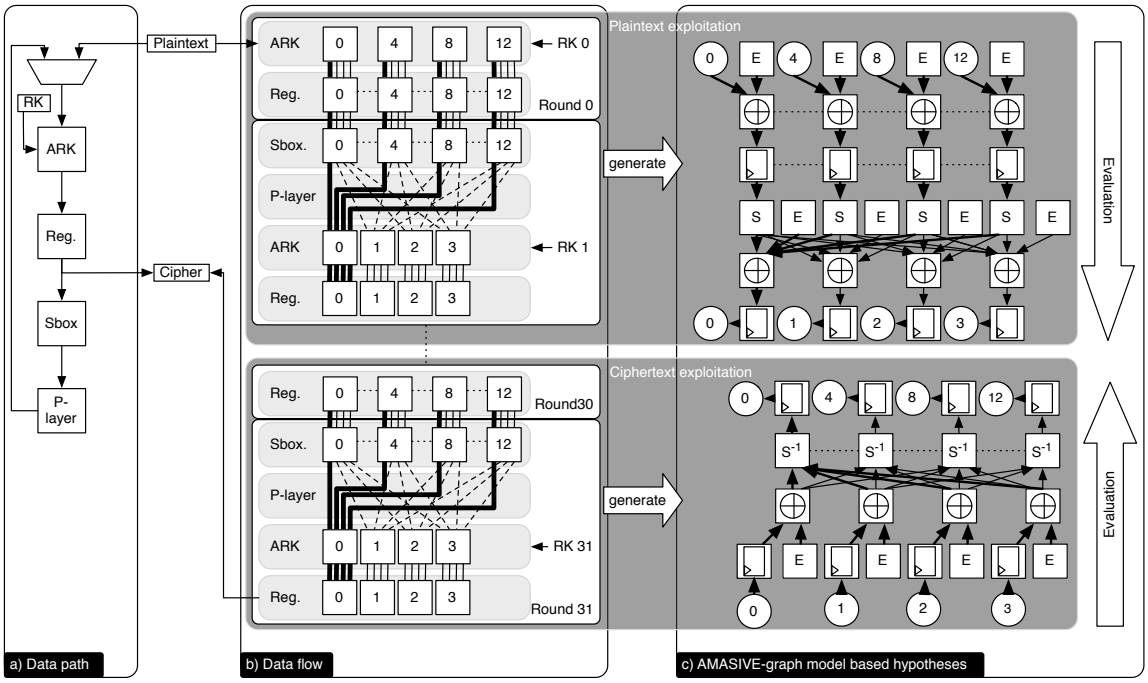

Figure 5. Part of the PRESENT block cipher represented as an analysis graph.

5.1.1. Side-Channel Analysis with CPA

Next, in order to rate the practicability of the attack, we mount a CPA using the hypothesis function ℓ on a SASEBO-GII FPGA implementation of the block cipher, which executes the outlined PRESENT. We performed 5000 measurements and used the Pearson correlation coefficient as the comparison method between the hypotheses and the power traces. The resulting correlation of the attack on the last round of PRESENT using ℓ confirms the correctness of the hypothesis. After 3400 captured traces, the complete round key was recovered, and thus, the secret key of the block cipher can be revealed with a computational effort of just 2^{16}.

5.2. Hardening the Unprotected PRESENT Version

After the analysis of the unprotected design and the verification by the conducted CPA, we now evaluate how well the AMASIVE framework can improve the design by automatically-embedded countermeasure schemes. According to Sections 3.1 and 4.2.1, the designer is free to apply any one or several ones from the set of available countermeasures, cf. Figure 4, and then to re-evaluate the hardened circuit by executing a CPA to check whether the selected countermeasures are sufficiently effective. Therefore, we investigate four cases with various side-channel hardened designs each. To be more specific, three out of the four new designs are hardened by using only one of the previously introduced countermeasures. The fourth hardened design version combines all three countermeasures of the previous designs. Hence, the ability of combining different countermeasure embedding processes of AMASIVE is investigated and presented as in the fourth case study.

5.2.1. Secure Synthesis based on Designer-controlled Countermeasure insertions

All components of the existing countermeasure plugins used for this case study were written as generic VHDL code and do not contain any specific usage of primitives, nor special placement and routing procedures on the FPGA fabric. Hence, the (re-)synthesis step may or may not provide an additional optimization of the design, which is performed by the logic synthesis tool of the FPGA programming tool chain.

The first hardened version exploits the hiding countermeasure random register switching to raise the effort of attacking the intermediate values of the state registers. This countermeasure needs in this case two additional registers and one 3:1 multiplexer in order to generate the non-deterministic switching behavior, cf. Figure 4. The random selection of the register of writing to and reading from is generated in this case by a commonly shared 16-bit LFSR with a non-public seed. Hence, the resource consumption of registers rises by an additional $(3 \cdot 64 + 16)$ registers, cf. Table 3.

Table 3. Resource consumption of the PRESENT hardened implementations.

	Resource Consumption					Performance	
	Register	**BRAM**	**LUT**	**Slices**	**Overhead (%)**	**Clock (MHz)**	**Throughput (MB/s)**
Unprotected	414	0	540	255	0	431	862
Random register switching	622	0	652	389	53	321	642
Masked register	607	0	749	312	22	381	762
Masked data path	624	0	842	339	32	415	830
Combined countermeasures	696	0	1036	401	57	256	512

The second investigated countermeasure uses the component masking plugin in order to mask the state register as visualized in the center part of Figure 4. Compared to the first countermeasure, the second one belongs to the class of masking and randomizes the internal computations by performing an XOR operation with a random value on the intermediate values. The random value is again generated by an LFSR with a non-public seed. The third countermeasure extending the design is a Boolean masking countermeasure as well, but now, it masks the entire data path, as shown in Figure 4 on the right-hand side. In this case study, the secure S-box component $S'(x)$ of the embedded countermeasure is based on the USM S-box scheme presented in [14]. We selected this scheme because it is the most generic and may be embedded into a data path featuring any non-linear logic operations.

As stated before, the last countermeasure is a combination of all of the previous countermeasures. Hence, the resulting design consists of randomly switching state registers, which are additionally masked in the Boolean masked data path, whereas the mask of the registers is different from the mask of the data path. Please keep in mind that the selection of countermeasures is under designer control whereas the subsequent embedding of these countermeasures is being done in a completely automatic manner by the advocated framework on top of the original VHDL code. Hence, the logic synthesis tool may further optimize the merged countermeasure-enhanced architecture, whereas the resource overhead of this case study is smaller than the sum of all three individual countermeasure designs, cf.

Table 3. However, due to resource optimization, the maximum clock frequency drops significantly compared to the previous design variants.

5.2.2. CPA of Hardened PRESENT Design Variants

We used the same input parameters (plain text) for all attacks, as well as the same secret. In total, we recorded 100,000 traces from each side-channel hardened design implemented running on the SASEBO-GII platform. We evaluated all four different side-channel hardened designs.

At first, we analyzed the simple hiding countermeasure random register switching. Compared to the unsecured version, this countermeasure slightly increased the resistance against a CPA attack using the leakage function ℓ, cf. Figure 6. However, still, all subkey nibbles were revealed, but with a ten-times higher effort due to the decreased signal-to-noise ratio caused by the hiding countermeasure, cf. Table 4. The second design variant with the masked register provides better resistance. Even using all 100,000 traces, only 12 out of the 16 subkeys were correctly revealed, which still leads at least to an effort of brute forcing 2^{28} possible subkey values in order to guess the remaining unknown key bits. The third design seems to be stronger than the second hardened design according to the graphic in Figure 6, because the best value of correctly-guessed subkeys is 11, but it is not really stable. A more stable value of the guess is 10 subkeys; hence, an attack has at least a computational effort of 2^{40} to brute force the remaining bits. The best protection while using just one countermeasure is thus offered by "data path masking". For the last design, only two subkeys were revealed by a CPA exploiting all traces, cf. Figure 6. Hence, in the worst case, the attack has to evaluate 2^{72} possible values in order to obtain the entire secret key, cf. Table 4.

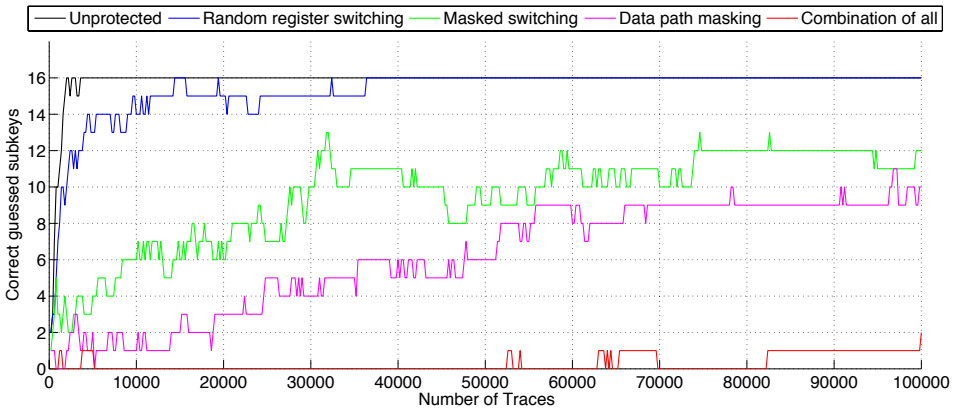

Figure 6. Correlation Power Analysis (CPA) of side-channel hardened PRESENT variants with hypothesis function ℓ.

The results of Table 4 should be interpreted as follows: the minimal number of traces for a certain number of correctly-found keys is not bounded to a specific subkey. In other words, if the number of correctly-found subkeys is 10 with 10,000 traces, for instance, and then drops and rises again to 10, but these correctly-found subkeys do not need to be the same subkeys as the previous ones. One can only be sure in considering that the same subkeys are correctly found if their number remains stable. Therefore, the numbers in Table 4 are estimates of the largest possible threat with the least amount of traces.

Table 4. Characteristics of side-channel attack of the PRESENT design variants.

Effort to Get by the Secret Key by Brute Force	Number of Correct Found Subkeys	Minimal Number of Needed Traces				
		Unprotect. Design	Hardened Design			
			Rand. reg. sw.	Masked reg.	Masked Data Path	Combined
2^{80}	0	0	0	0	0	0
2^{76}	1	5	9	14	25	34
2^{72}	2	6	38	140	55	100,000
2^{68}	3	22	112	263	2223	>100,000
2^{64}	4	85	162	321	22,194	>100,000
2^{60}	5	137	437	565	24,527	>100,000
2^{56}	6	423	475	705	27,660	>100,000
2^{52}	7	449	732	8511	35,811	>100,000
2^{48}	8	527	957	11,330	51,154	>100,000
2^{44}	9	543	1185	22,560	55,168	>100,000
2^{40}	10	645	1274	27,472	77,785	>100,000
2^{36}	11	664	1342	29,835	96,480	>100,000
2^{32}	12	688	2112	30,561	>100,000	>100,000
2^{28}	13	1358	2269	31,066	>100,000	>100,000
2^{24}	14	1381	4258	>100,000	>100,000	>100,000
2^{20}	15	1497	6016	>100,000	>100,000	>100,000
2^{16}	16	1548	11,824	>100,000	>100,000	>100,000

5.2.3. Discussion of Analysis Results

The evaluation demonstrates that the AMASIVE framework is in general well-suited to automatically embed effective countermeasures against side-channel attacks into a given design coded in VHDL and thereby increases significantly the required attacking effort compared to an unprotected device. In addition, the resource vs. security trade-off for side-channel hardening can be adjusted according to the security scenario at hand. The strengths against side-channel attacks, in this case CPA of the dynamic power consumption, strongly depend on the primitive components, which are provided within the framework library due to its plugin structure. The same holds for the resource consumption of the countermeasure-enhanced generated design. The key feature to gain this flexibility in the design flow is the introduction of the (re-)synthesis concept, which on the other hand may lead in some cases to new side-channel flaws due to the optimization setting for the subsequent logic synthesis step or during the place and route procedure. An optimization may be avoided for certain entities, i.e., the hardened ones, by assigning "don't touch" attributes to them, a feature supported by most logic synthesis tools. Hence, the responsibility of the designer to implement adequate side-channel secured components is still present, but it is now more generalized, so that generic components can be reused for different circuits.

The impact on the side-channel resistance of the primitive components, which are available within AMASIVE for the purpose to generate the countermeasure, is clearly reflected in the results of the CPA analysis. For instance, the effect of the hiding countermeasure denoted as random register switching is not meeting the a-priori expectation: The increased effort was roughly 10 times, because when providing three registers to store a value, the probability of sensing a transition on the same register is $p(r_i \rightarrow r_i) = \frac{1}{9}$.

In contrast, the countermeasure named masked register using the USM scheme is not as side-channel resistant as expected. The masking scheme of the data path is still exploitable due to two facts:

- The scheme's core component, i.e., the shared/masked S-box, is not entirely implemented by using the primitive components of the platform (Virtex 5 in this case).
- The USM masking scheme was used based on its generic structure, but not realized by exploiting a Block-RAM module. Hence, the correction part of the USM is unmasked, cf. [14], and additionally prone to data-dependent glitches, as discussed in [30].

This example demonstrates the importance of a proper implementation of the side-channel hardened primitive components. In the case of the masking scheme, a superior method may be the TI

scheme presented in [12], which is resistant against glitches. It was already demonstrated in [31,32] that this scheme is successfully applicable to PRESENT in order to gain first order side-channel resistance against CPA, but the approach is until now not as generally applicable, such as the traditional Boolean masking. We favored the latter method for the following reasons: For each application, the masking scheme has to be adapted to the specific S-box of the applied cipher, which works for all S-box variants up to four bits only, cf. [32,33]. Even for five bits, the search space for a suitable solution becomes so large that it is not traversable in a realistic search time. Hence, the TI masking scheme is in terms of the presented concept not sufficiently modular and general when compared to Boolean masking on top of USM, but it may still be applicable.

6. Conclusions

The AMASIVE framework aims at efficiently supporting a designer in hardening a hardware implementation of a cryptographic algorithm against side-channel power attacks. On the one hand, the framework autonomously identifies the weaknesses of a given design description. On the other hand, AMASIVE offers appropriate generic countermeasures based on the discovered weaknesses and integrates them into a hardened design variant. In this paper, we focused on the set of currently-available countermeasures and on the method for how they can be automatically inserted into a design. The proposed framework represents the cryptographic algorithm during its workflow as a dedicated graph and exploits a sophisticated and adaptable attacker model in order to determine side-channel vulnerabilities in the original design. Compared to the rather few published approaches to produce secure hardware designs, this framework is very flexible in terms of both the attack method and the application of the countermeasure regarding the technology platform of the target device. Different countermeasure techniques taken from the fundamental classes known as hiding and masking are integrated via a data base into the framework. The resulting security building blocks are then autonomously used by the framework to construct a side-channel hardened version of the original circuit without requiring the designer to write any additional line of functional HDL code.

We evaluated our concept of security-driven (re-)synthesis and the resulting analysis and design framework by means of a completely implemented encryption module of the block cipher PRESENT in order to demonstrate that the framework can handle complex circuit structures instead of just isolated operational components needed in a cryptographic algorithm like an S-box. First, the framework automatically analyzed the original version of the given PRESENT VHDL code in order to produce attacking vectors in form of side-channel hypotheses. Then, four different, secured versions of the original block cipher circuit were generated by the advocated security-driven synthesis approach and then evaluated by means of a CPA. These design variants were subsequently compared in terms of the resource consumption of the applied countermeasure and of the additional effort to mount a side-channel attack on the hardened circuit. In doing so, the scalability of securing a given implementation was demonstrated.

In contrast to existing side-channel security approaches, we focus on adaptability and extensibility in order to support the analysis and the resistance improvement of a wide variety of crypto-systems. In future work, we will extend the constructive part of this design system by more sophisticated countermeasures. In addition, we will considerably extend the capabilities of the attacker by introducing new attack and evaluation methods, such as the template attack and the mutual information metric. Moreover, we aim to extend and to apply the AMASIVE framework to additional cryptographic algorithms.

Acknowledgments: The work presented in this paper was supported in part by the German Federal Ministry of Education and Research (BMBF) in the joint project RESIST under Grant Number 01IS10027A and by the CASED research center. The authors extend their thanks to Marc Stöttinger and Michael Zohner for their contributions to this project and to the anonymous reviewers for their valuable comments and suggestions.

Author Contributions: The authors shared the work related to both the development of the fundamental concept and to the writing of this paper.

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

References

1. Brier, E.; Clavier, C.; Olivier, F. Correlation Power Analysis with a Leakage Model. In *Series Lecture Notes in Computer Science, Proceedings of the 6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004), Cambridge, MA, USA, 11–13 August 2004*; Joye, M., Quisquater, J.-J., Eds.; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3156, pp. 16–29.
2. Le, T.; Canovas, C.; Clediere, J. An Overview of Side Channel Analysis Attacks. In Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security (ASIACCS 2008), Tokyo, Japan, 18–20 March 2008; ACM Digital Library: New York, NY, USA, 2008.
3. Mangard, S.; Popp, T.; Oswald, M.E. *Power Analysis Attacks—Revealing the Secrets of Smart Cards*; Springer: New York, NY, USA, 2007.
4. Veyrat-Charvillon, N.; Medwed, M.; Kerckhof, S.; Standaert, F.-X. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *Series Lecture Notes in Computer Science, Proceedings of the 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, 2–6 December 2012*; Wang, X., Sako, K., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7658, pp. 740–757.
5. Cohen, H.; Frey, G. (Eds.) *Handbook of Elliptic and Hyperelliptic Curve Cryptography*; CRC Press: Boca Raton, FL, USA, 2005.
6. Tiri, K.; Verbauwhede, I. A Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Paris, France, 16–20 February 2004; IEEE: Washington, DC, USA, 2004; pp. 246–251.
7. He, W.; de la Torre, E.; Riesgo, T. An Interleaved EPE-Immune PA-DPL Structure for Resisting Concentrated EM Side Channel Attacks on FPGA Implementation. In *Series Lecture Notes in Computer Science, Proceedings of the Third International Workshop on the Constructive Side-Channel Analysis and Secure Design (COSADE 2012), Darmstadt, Germany, 3–4 May 2012*; Schindler, W., Huss, S.A., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7275, pp. 39–53.
8. Lu, Y.; Boey, K.; Hodgers, P.; O'Neill, M. Lightweight DPA Resistant Solution on FPGA to Counteract Power Models. In Proceedings of the 2010 International Conference on Field-Programmable Technology (FPT), Beijing, China, 8–10 December 2010; Bian, J., Zhou, Q., Athanas, P., Ha, Y., Zhao, K., Eds.; IEEE: Washington, DC, USA, 2010; pp. 178–183.
9. Güneysu, T.; Moradi, A. Generic Side-Channel Countermeasures for Reconfigurable Devices. In *Series Lecture Notes in Computer Science, Proceedings of the 13th International Workshop, Nara, Japan, 28 September–1 October 2011*; Preneel, B., Takagi, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6917, pp. 33–48.
10. Lu, Y.; O'Neill, M.; McCanny, J.V. Evaluation of Random Delay Insertion against DPA on FPGAs. *TRETS* **2010**, *4*, 11.
11. Nikova, S.; Rechberger, C.; Rijmen, V. Threshold Implementations Against Side-Channel Attacks and Glitches. In *Series Lecture Notes in Computer Science, Proceedings of the 8th International Conference on Information and Communications Security (ICICS 2006), Raleigh, NC, USA, 4–7 December 2006*; Ning, P., Qing, S., Li, N., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4307, pp. 529–545.
12. Nikova, S.; Rijmen, V.; Schläffer, M. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptol.* **2011**, *24*, 292–321.
13. Chari, S.; Jutla, C.S.; Rao, J.R.; Rohatgi, P. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *Series Lecture Notes in Computer Science, Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO '99), Santa Barbara, CA, USA, 15–19 August 1999*; Wiener, M.J., Ed.; Springer: Berlin/Heidelberg, Germany, 1999; Volume 1666, pp. 398–412.
14. Maghrebi, H.; Danger, J.-L.; Flament, F.; Guilley, S.; Sauvage, L. Evaluation of Countermeasure Implementations Based on Boolean Masking to Thwart Side-Channel Attacks. In *Proceedings of the 2009 3rd International Conference on Signals, Circuits and Systems (SCS), Medenine, Tunisia, 6–8 November 2009*; IEEE: Jerba, Tunisia, 2009.

15. Popp, T.; Mangard, S. Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints. In *Series Lecture Notes in Computer Science, Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005), Edinburgh, UK, 29 August–1 September 2005*; Rao, J.R., Sunar, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3659, pp. 172–186.
16. Popp, T.; Kirschbaum, M.; Zefferer, T.; Mangard, S. Evaluation of the Masked Logic Style MDPL on a Prototype Chip. In *Series Lecture Notes in Computer Science, Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), Vienna, Austria, 10–13 September 2007*; Paillier, P., Verbauwhede, I., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4727, pp. 81–94.
17. Bayrak, A.G.; Regazzoni, F.; Brisk, P.; Standaert, F.-X.; Ienne, P. A first step towards automatic application of power analysis countermeasures. In *Proceedings of the 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC), San Diego, CA, USA, 5–9 June 2011*; Stok, L., Dutt, N.D., Hassoun, S., Eds.; ACM: New York, NY, USA, 2011; pp. 230–235.
18. Standaert, F.-X.; Malkin, T.; Yung, M. A unified framework for the analysis of side-channel key recovery attacks. In *Series Lecture Notes in Computer Science, Proceedings of the 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, 26–30 April 2009*; Joux, A., Ed.; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5479, pp. 443–461.
19. Oswald, E.; Mather, L.; Whitnall, C. Choosing distinguishers for differential power analysis attacks. In Proceedings of the Non-Invasive Attack Testing Workshop, Nara, Japan, 25–27 September 2011.
20. Whitnall, C.; Oswald, E. A Fair Evaluation Framework for Comparing Side-Channel Distinguishers. *J. Cryptogr. Eng.* **2011**, *1*, 145–160.
21. Whitnall, C.; Oswald, E. A Comprehensive Evaluation of Mutual Information Analysis Using a Fair Evaluation Framework. In *Series Lecture Notes in Computer Science, Proceedings of the 31st Annual Cryptology Conference on Advances in Cryptology (CRYPTO 2011), Santa Barbara, CA, USA, 14–18 August 2011*; Rogaway, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6841, pp. 316–334.
22. Moss, A.; Oswald, E.; Page, D.; Tunstall, M. Automatic insertion of dpa countermeasures. *IACR Cryptol. ePrint Arch.* **2011**, *2011*, 412.
23. Moss, A.; Oswald, E.; Page, D.; Tunstall, M. Compiler assisted masking. In Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2012), Leuven, Belgium, 9–12 September 2012; pp. 58–75.
24. Regazzoni, F.; Cevrero, A.; Standaert, F.-X.; Badel, S.; Kluter, T.; Brisk, P.; Leblebici, Y.; Ienne, P. A design flow and evaluation framework for dpa-resistant instruction set extensions. In *Series Lecture Notes in Computer Science, Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2009), Lausanne, Switzerland, 6–9 September 2009*; Clavier, C., Gaj, K., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5747, pp. 205–219.
25. Bayrak, A.G.; Regazzoni, F.; Novo, D.; Brisk, P.; Standaert, F.-X.; Ienne, P. Automatic Application of Power Analysis Countermeasures. *IEEE Trans. Comput.* **2015**, *64*, 329–341.
26. Elaabid, M.A.; Guilley, S. Practical improvements of profiled side-channel attacks on a hardware crypto-accelerator. In *Series Lecture Notes in Computer Science, Proceedings of the 8th International Conference on Cryptology in Africa, Fes, Morocco, 13–15 April 2016*; Bernstein, D.J., Lange, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6055, pp. 243–260.
27. Zohner, M.; Stöttinger, M.; Huss, S.A.; Stein, O. An adaptable, modular, and autonomous side-channel vulnerability evaluator. In Proceedings of the 2012 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), San Francisco, CA, USA, 3–4 June 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 43–48.
28. Bogdanov, A.; Knudsen, L.R.; Leander, G.; Paar, C.; Poschmann, A.; Robshaw, M.J.B.; Seurin, Y.; Vikkelsoe, C. Present: An ultra-lightweight block cipher. In *Series Lecture Notes in Computer Science, Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2007), Vienna, Austria, 10–13 September 2007*; Paillier, P., Verbauwhede, I., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4727, pp. 450–466.
29. Huss, S.A. *RESIST—Methods and Tools to Protect Embedded and Mobile Systems against Next-Generation Attacks*; Final project report; ICS Lab, Technische Universität Darmstadt: Darmstadt, Germany, 2013. (In German)

30. Fischer, W.; Gammel, B.M. Masking at Gate Level in the Presence of Glitches. In *Series Lecture Notes in Computer Science, Proceedings of the 7th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2005), Edinburgh, UK, 29 August–1 September 2005*; Rao, J.R., Sunar, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3659, pp. 187–200.
31. Poschmann, A.; Moradi, A.; Khoo, K.; Lim, C.-W.; Wang, H.; Ling, S. Side-Channel Resistant Crypto for Less than 2, 300 GE. *J. Cryptol.* **2011**, *24*, 322–345.
32. Kutzner, S.; Nguyen, P.H.; Poschmann, A.; Wang, H. On 3-Share Threshold Implementations for 4-Bit S-boxes. In *Series Lecture Notes in Computer Science, Proceedings of the 4th International Workshop (COSADE 2013), Paris, France, 6–8 March 2013*; Prouff, E., Ed.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7864, pp. 99–113.
33. Bilgin, B.; Nikova, S.; Nikov, V.; Rijmen, V.; Stütz, G. Threshold Implementations of All 3×3 and 4×4 S-Boxes. In *Series Lecture Notes in Computer Science, Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems (CHES 2012), Leuven, Belgium, 9–12 September 2012*; Prouff, E., Schaumont, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7428, pp. 76–91.