# Chapter 8
# A Novel Mutating Runtime Architecture for Embedding Multiple Countermeasures Against Side-Channel Attacks

**Sorin A. Huss and Marc Stöttinger**

## 8.1 Introduction

In today's modern communication technology-based society embedded electronic devices get more and more into our daily life. The purposes of these embedded systems are quite different, but most of them require to transfer data between components within a device or between various devices. At least part of the transferred information is sensitive and thus has to be protected in order to avoid eavesdropping and misuse. In principle this problem may be solved by means of cryptographic algorithms implemented in software. However, the related overhead such as additional power consumption needs to be considered carefully. A more appropriate solution to achieve the required computational performance is to introduce dedicated microelectronic modules to efficiently speed-up these operations. One option is to use FPGA-based ciphers to improve the system performance without losing the important software-like flexibility.

In 1999 new attack methods known as Simple Power Attack (SPA) and Differential Power Analysis (DPA) have been introduced [9], which since then significantly changed the view on design requirements of security sensitive applications. An adversary just needs to observe the physical behavior of the cryptographic module, while it operates in its normal mode and then to exploit the recorded data by analyzing key-dependent properties such as execution time, power consumption,

S.A. Huss (✉)
CASED - Center of Advanced Security Research Darmstadt, and Integrated Circuits and Systems Lab, Computer Science Department, Technische Universität Darmstadt, Hochschulstrasse 10, Darmstadt, Germany
e-mail: huss@iss.tu-darmstadt.de; sorn.huss@cased.de

M. Stöttinger
Independent Researcher (formerly with CASED), Darmstadt, Germany
e-mail: marc.stoettinger@gmail.com

P. Mishra et al. (eds.), *Hardware IP Security and Trust*,
DOI 10.1007/978-3-319-49025-0_8

or electromagnetic radiation. Such passive, non-invasive characteristics make side-channel attacks (SCA) rather dangerous because they are efficient, easy to conduct, and do not leave any evidence of the attempted assault. Especially the power analysis attack is one of the most popular classes of passive SCA strategies and is therefore being thoroughly investigated in academia as well as in industry.

Many countermeasures have meanwhile been proposed aimed to hardening implementations of cryptographic modules against such attacks. The main goal of hardening a design by introducing countermeasures is to reduce and, in the best case, to minimize the amount of exploitable physical information. Thus, next to modifying an encryption algorithm, the envisaged technology platform as well as the architectural structure of the module needs to be adapted accordingly.

Countermeasures against power analysis attacks are mainly applied on the algorithm or on the cell level of the implementation platform. The reason for this fact is that countermeasures are bound directly to the envisaged platform. This means that in case of a software implementation the countermeasures can only be mapped to software as part of the cryptographic algorithm, i.e., on algorithmic level. Masking is a good example for such a countermeasure. In case of exploiting a custom chip as platform, the designer may apply circuit-related countermeasures on cell level such as Dual-Rail logic. Please note that in both cases the physical architecture of the implementation is not being modified. In contrast, the concept of an FPGA platform offers much more flexibility. This is because of the inherent reconfiguration capability of both functional blocks and their interconnection on an FPGA—even at runtime.

In the context of countermeasures, however, only one approach has been published so far which directly addresses the architecture level by exploiting the concept of modifying the physical structure of the platform. The so-called Temporal Jitter method proposed by Mentens et al. in [14] exploits the reconfiguration capability of the data path in order to freely position both registers and combinatoric logic in the circuit. In doing so, the activity pattern of each clock cycle is being modified and, similar to Shuffling on algorithmic level, the adversary cannot directly deduce from the recorded power traces which specific operation is being processed. Platform reconfiguration is therefore the key method to accommodate on all abstraction levels highly variable, at runtime mutating architectures as addressed in this chapter.

The paper is structured as follows. Section 8.2 introduces the basic concepts, multiple countermeasures resulting from a tailored digital system design methodology, and the advocated architecture. Section 8.3 summarizes the design flow of the Mutating Runtime Architecture. In Sect. 8.4 we detail the various design decisions and demonstrate the advantages of the proposed approach by hardening an AES block cipher. Finally, Sect. 8.5 concludes the chapter.

## 8.2 Mutating Runtime Architecture

The ongoing research in the area of computer-aided engineering (CAE) has meanwhile developed various techniques to map a formally denoted algorithm to an integrated circuit design. High-level synthesis is one of the core concepts in such a design flow. It consists of the following fundamental design activities in order to implement an algorithm in hardware: Allocation, Binding, and Scheduling.

First, the elementary operations needed to execute the algorithm are identified during Allocation. Each such operation is then mapped to one or two multiple types of suited hardware processing units. An explicit assignment of a specific task of the algorithm to a unit is addressed by Binding. Finally, Scheduling is required to order tasks in the time domain. Thus, by performing this activity the designer decides on the number of operations to be executed in parallel or on the sequence of data-independent operations. The resulting architectural properties of the circuit rely directly on the results of the above-mentioned basic activities of the CAE process, cf. [6]. Thereby, physical characteristics of the implemented algorithm may be affected significantly. Please note that design decisions in one of these activities in general do influence decisions in the other ones as well.

### *8.2.1 Design Properties*

The concept of the Mutating Runtime Architecture (MRA) exploits the intrinsic influence of the mentioned design activities on the architecture in order to enable a flexible design space exploration. This exploration results in various feasible implementations of the same cryptographic algorithm, but with quite different physical properties. The challenge resulting from this approach is to identify the fundamental design properties which can be controlled individually by the corresponding design activities.

The outcome of Allocation specifies which *types of processing units* are required by the algorithm and thus denotes the first design property. Type of processing unit inherits its parameters from Allocation, but it is nearly independent from Binding and Scheduling. Thus, it is possible to replace an individual processing unit at runtime by means of a novel method named OnlineAllocation. The second property is the *degree of parallelism*, which inherits its design parameters from Binding. Independently from the type of the allocated processing unit, this degree can be varied based on the amount of available units. However, it can only be modified if the mutating method named DynamicBinding detailed in the sequel does not violate task dependencies. The last property is linked to Scheduling and addresses the *progression of tasks*, which are to be executed sequentially in a data-dependent order. The data flow is independent from the utilized processing units from a functional point of view. The related method FlexibleScheduling thus deals with task progression, whereas the overall task management shall be addressed by a middleware approach. Figure 8.1 visualizes the relationships of design activities, properties, and dedicated countermeasure construction methods.

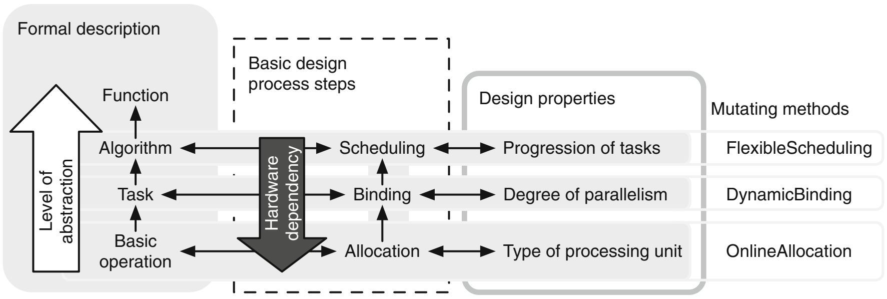


**Fig. 8.1** Layer model for design activities, properties, and mutating methods

Hardware/Software co-design seems to be a feasible fundamental approach according to the discussion outlined above. Both task progression management and dynamic binding for the selected degree of parallelism are much easier to handle in software. Only the online allocation of different types of processing units cannot be assigned to the software partition due to their inherent hardware dependency thus resulting in a tailored HW/SW architecture on top of the envisaged FPGA implementation platform.

### *8.2.2 Online Allocation Method*

The overall power consumption of CMOS circuits results to a large extent from the switching activity of the circuit. More specifically speaking, the various internal modules, interconnections, and configuration settings of LUTs of the FPGA platform are the origin of the characteristic controlled transient effects or glitches. Therefore, different implementations of the same basic operation within a cryptographic algorithm clearly influence the exploitable power consumption signature because of their also distinct switching activities. The statistical properties of the data-dependent power consumption in terms of mean and variance are thereby strongly affected by the characteristics of the glitches of the circuit. In case of a static design this property is relatively vulnerable to passive side-channel attacks, cf. [5, 11, 12, 21]. These attacks exploit the unique distribution of the power consumption signature. One possibility to compare the power consumption signature of different design variants featuring the same functionality is to produce a scatter plot like in Fig. 8.2.

In this figure we visualize the power consumption of four different designs of an SBox operation. Each of these circuits consists of one 8-bit register and one of the four different 8-bit AES SBox designs known as COMP, PPRM1, PPMR3, and TBL, respectively. On the $x$-axis the power consumption value of SBox design A at point in time $t$ is displayed. The $y$-axis marks the related value of SBox design B. This example clearly indicates the difference in the exploitable power consumption

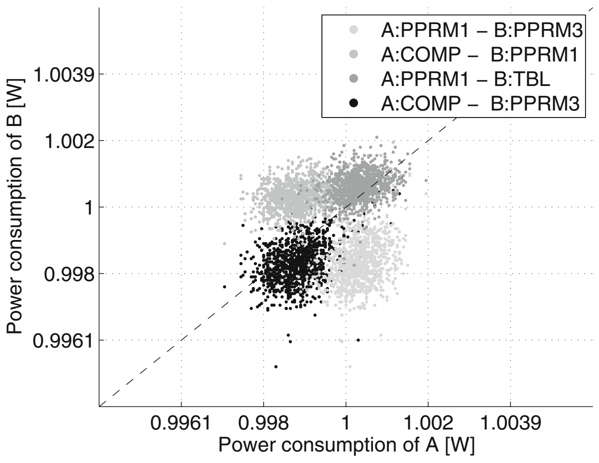


**Fig. 8.2** Scatter plot of the power consumption of different SBox design variants

signature of design variants. Both variant-specific characteristics, i.e., the LUT configuration and the interconnect architecture of the LUT on the FPGA platform, cause different distributions of the power consumption, whereas the distribution strongly depends on the switching activity. This behavior may well be exploited as a countermeasure because processing units featuring identical functionality but different signatures help to minimize the exploitable power characteristics. However, one has to take into account that a low overall total correlation value between variants is a necessary, but not a sufficient condition to generate a complex distribution based on switching randomly between the individual distributions of different types of units. A more appropriate metric is the total correlation value of Hamming distance classes between different design variants of the same processing unit as proposed in [19].

The FPGA technology offers many techniques to take the concept of online allocation to practice. So, the exchange of processing unit types may be applied at various places. First, the randomization can be done within the switching network. Second, it may be applied on platform level by exploiting the partial reconfiguration features of an FPGA. These techniques differ in the required amounts of resource consumption and total power consumption as well as of design time due to the resulting device complexity and the available tool support. One of the first approaches to replace active units in order to manipulate the overall power consumption signature is reported in [1, 2]. There, Benini et al. proposed to utilize two different implementations of a processing unit and to randomly switch between them by means of a multiplexor. Compared to this work, these authors do not quantify the power characteristics in terms of statistical properties.

**Switching Network** A straight-forward approach to reshape the data path at runtime is to exploit a switching network based on multiplexors. The resulting advantage is the small amount of time needed to mutate the data path behavior in terms of its dynamic power consumption. The main drawbacks, however, result from a considerable enlargement of the data path depth and from a decreased throughput. A substantial improvement in terms of resource consumption and throughput is presented in [10]. Here, instead of switching between two complete processing units to mutate the data path behavior, the randomization addresses directly the elementary components of the units. A lot of resources on the primitive layer of the FPGA platform can thus be shared and the unavoidable data path depth incrementation is under better control. To reach this goal, Madlener et al. in [10] propose a novel hardened multiplier for GF($2^n$) aimed to public-key ECC applications denoted as “enhanced Multi-Segment-Karatsuba” (eMSK), which is an improvement of the Multi-Segment Karatsuba (MSK) multiplier introduced in [4]. The main advantage of the eMSK scheme is that it combines the efficiency of the recursive Karatsuba with the flexibility of the MSK scheme. Therefore, the order of the operations in the sequence strongly affects the power consumption due to a randomization in both time and amplitude domain resulting in an efficient countermeasure. Consequently, the varying order of the basic multiplication has a strong impact on the glitches in the combinatorial multiplier logic as well as on the bit flips in the accumulator. A similar countermeasure applied to the AES block cipher is proposed in [7], where the technique of composite-field-based SBoxes is exploited in order to switch between different variants. Such a concept was first investigated in [3]. Compared to many other masking schemes such as of, e.g., [17], the data path in [7] does not need to be doubled thus making this countermeasure suitable even for resource constrained designs.

**Partial Reconfiguration** Some RAM-based FPGA platforms such as Xilinx Virtex 5 support dynamic partial reconfiguration, a feature which may be exploited for the purpose of mutating data paths too. Instead of using more active reconfigurable resources for processing units operating in parallel, partial reconfiguration can be applied to save on resources. The data path depth increases not that much compared to a switching network and the resulting total power consumption is lower.

### *8.2.3 DynamicBinding Method*

The property *degree of parallelism* influences the time-dependent amount of noise due to concurrently operating processing units. Therefore, the variance of the power consumption at a certain point in time is being manipulated. Variance manipulation strongly compromises the learning phase of a template attack, in which an attacker tries to establish a multivariable Gaussian-based model for the noise distribution. In addition, a dynamic activation of several processing units working in parallel also affects the noise characteristics of the measured power consumption. One efficient method to provide a dynamic binding of resources stems from the concept of virtualizing hardware components. Virtualization may be seen as an abstraction layer, which changes during runtime the link between a processing unit and an assigned task. Therefore, different tasks, which are bound to one or two multiple

**Algorithm 8.3:** RanParExec algorithm

```
Require: a given set of n processing units of m different types $PU_{set} = \{PU_1^{types}, PU_2^{types}, \ldots, PU_n^{types}\}$ with $PU_k^{types} = \{PU^1, PU^2, \ldots, PU^m\}$, $1 \leq k \leq n$ and two equal distributed random variables $R_1 = \{1, 2, \ldots, m\}$, $R_2 = \{1, 2, \ldots, n\}$

 1: for $1 \leq w \leq u$ do
 2:     select a realization of the random variable $r_{1,w} \leftarrow R_1$
 3:     assign each $op_{PU,w}$ to the processing unit type $PU^{r_{1,w}}$
 4: end for
 5: repeat
 6:     repeat
 7:         select a realization of the random variable $r_{2,w} \leftarrow R_2$
 8:     until $r_{(2,w)} \leq$ number of currently in parallel executing $op_{PU,w+}$

 9:     Execute the following k operations in parallel
10:     for $1 \leq k \leq r_{(2,w)}$ do
11:         execute $op_{PU,k}$ with the assigned processing Unit PU
12:     end for
13:     $w = w + r_{(2,w)}$
14: until $w > u$
15: return
```

resources, can be shared or reordered. This kind of context switching on a processing unit may be handled transparently and does not need any modification of the scheduled task sequence within a cryptographic algorithm.

**Random Concurrent Binding** In this approach the number of units running in parallel varies as well as the binding of an operation to a specific unit. Therefore, several random combinations of unit types as provided by the method OnlineAllocation featuring different degrees of parallelism and bindings may thus be generated. Algorithm 8.3 named RanParExec illustrates this randomization technique denoted in pseudocode.

**Virtualization** Hardware virtualization is a method to abstract and decouple the executing hardware platform from the allocated algorithms. The virtualization technique presented in [20] exploits hardware components of the underlying FPGA platform and is taken as the basis for the approach of this work. A virtualized component may be shared by one or more tasks of a procedure or even of a complete algorithm. This component is addressed individually by the utilizing task set. But this task set allocates the hardware resource without any knowledge on whether it is being shared with a disjoint set of tasks.

**Middleware Concept** The application of virtualization between the executing processing units and the sequence of tasks based on the control and data flow can be established by an intermediate layer. It features almost the same functionalities as the so-called Middleware, a commonly used concept in the distributed computing domain. Thus, this concept may be viewed as an additional executive layer to

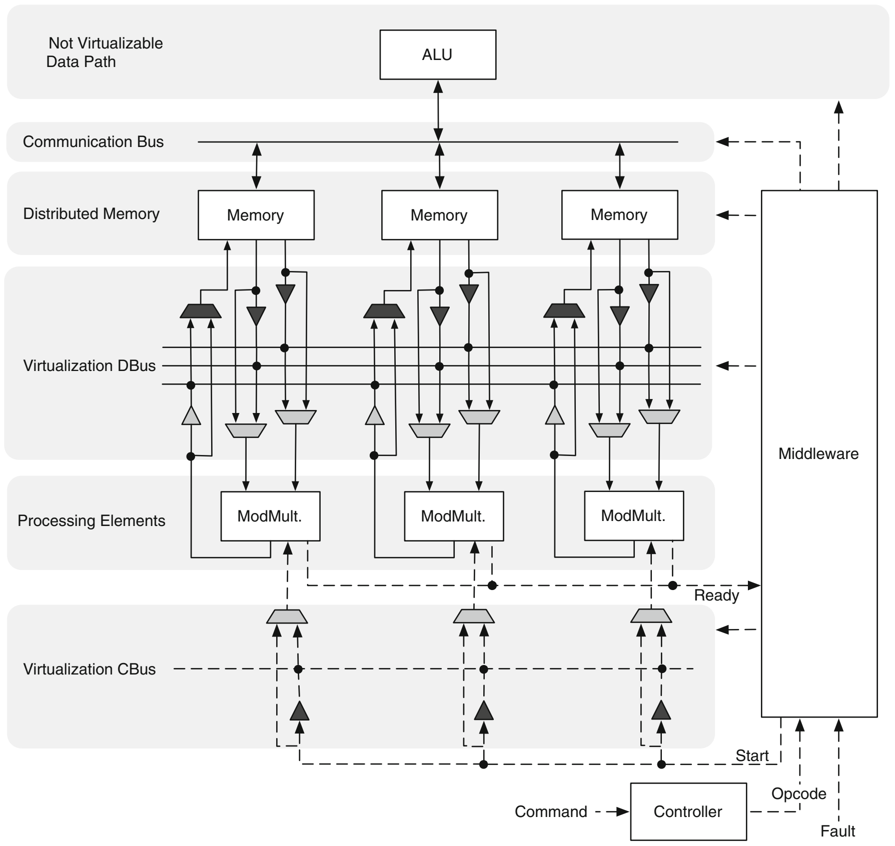
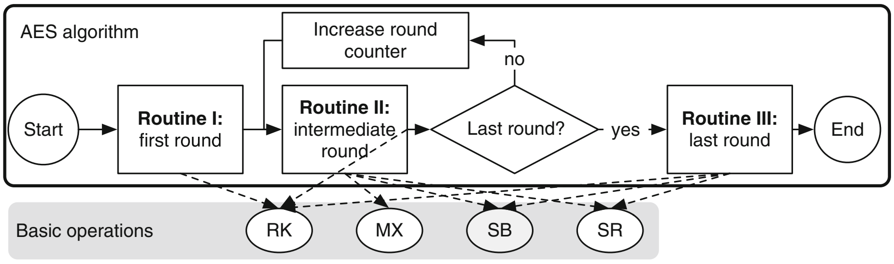


**Fig. 8.3** Virtualized public-key ECC cipher architecture [20]

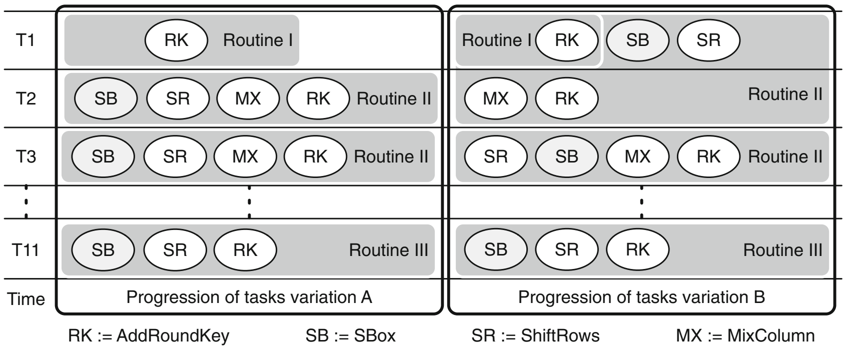
apply the method of virtualization for the outlined DynamicBinding method and has the administrative duty to organize the required links between tasks and executing resources. Figure 8.3 illustrates the outlined combined hardware virtualization and middleware approach applied to an ECC cipher architecture.

### *8.2.4 FlexibleScheduling Method*

The last design property of Fig. 8.2 named *progression of tasks* is being addressed by FlexibleScheduling, which is the final method introduced to further enhance the MRA by additional countermeasures. FlexibleScheduling is thus intended to directly change the execution behavior of the tasks within the cryptographic algorithm during runtime, which affects considerably the physical behavior of the implementation. The impact on the power consumption signature due to progression



(a) Control flow of the AES algorithm



(b) Example of rearranged basic operations in a routine

**Fig. 8.4** Rearrangement of operations by routine (re)shaping within AES. (**a**) Control flow of the AES algorithm. (**b**) Example of rearranged basic operations in a routine

of tasks is comparable to the commonly used method of shuffling independent operations in a cryptographic algorithm, if the granularity of processed operations per clock cycle does not change. Therefore, FlexibleScheduling decomposes a routine such that only the sequence of data-independent basic operations within the routine is being manipulated. The increased analysis effort for attacking a cryptographic module, which shuffles its data-independent operations, is well known [13]. Some interesting effects of FlexibleScheduling are illustrated in Fig. 8.4.

Most cryptographic algorithms are constructed either from basic operations or exploit other cryptographic functions, which are in turn composed of such operations. For instance, most block cipher algorithms such as AES or PRESENT are round oriented schemes. As illustrated in Fig. 8.4a, each different round function of AES can be implemented in its own routine, which utilizes a set of basic operations. Each of the three routines can be manipulated in terms of its physical behavior by means of different basic operations executed in one clock cycle without needing to change neither the number of bits processed in parallel nor the circuit-level implementation of these operations. In this example the basic operation of the SBox is handled by a mutable processing unit, which can be reallocated online while the device is active. In addition, the number of SBoxes operating in parallel may be modified by a dynamic binding. As the rearrangement example in Fig. 8.4b demonstrates, the first round of AES, which is one of the two most common targets of a power analysis attack, can be merged with some basic operations of the second one. Therefore, the number of operations executed within one clock cycle is modified from the original scenario depicted on the left-hand side of Fig. 8.4b to the rearranged routine shown on the right-hand side. The power consumption signature is being disturbed by the sum of Gaussian distributions caused by operations being active during one clock cycle over the observed set of experiments featuring a varying operational activity. Thus, the resulting complex joint distribution acts as an additional countermeasure because it combines effects stemming both from manipulations by means of DynamicBinding and from the exchange of unit types resulting from OnlineAllocation.

## 8.3 Design Flow

The overall design flow for a Mutating Runtime Architecture results from the outlined design strategy and is illustrated in Fig. 8.5. Starting from a given cryptographic algorithm, a HW/SW partitioning needs to be done first such that procedures dominating the control flow of the algorithm are assigned to the software partition and the basic operations are grouped to form the hardware partition. Tasks composed mainly from basic operations should be mapped to software too in order to better utilize the hardware resources in the data path.

The information clustered in the software partition forms the basis for establishing various schedules by utilizing FlexibleScheduling. This method can be used to decompose routines or to rearrange procedures within a routine too. In contrast, OnlineAllocation and DynamicBinding address the hardware partition. First, the basic operation executed on a processing unit with the most exploitable power consumption $\mathrm{op}_{\mathrm{PU}}$ needs to be identified. The units aimed to run the other basic operations can be implemented directly on the target FPGA as depicted in the lower part of Fig. 8.5. The generated schedules are then analyzed in order to identify the amount of $\mathrm{op}_{\mathrm{PU}}$ usage. After this step, DynamicBinding produces the virtualization schemes. They are utilized in the sequel to construct the related virtualization module. The processing of the schemes for OnlineAllocation, FlexibleScheduling, and DynamicBinding can be conducted concurrently.

In the end, after these construction methods have jointly generated multiple countermeasures, all required components are assembled into the MRA featuring a virtualization module and various types of processing units. The randomization of the active processing units is handled directly by the middleware, i.e., the virtualization module, via a switching network or, if applicable, by exploiting a partial reconfiguration of the target FPGA platform.

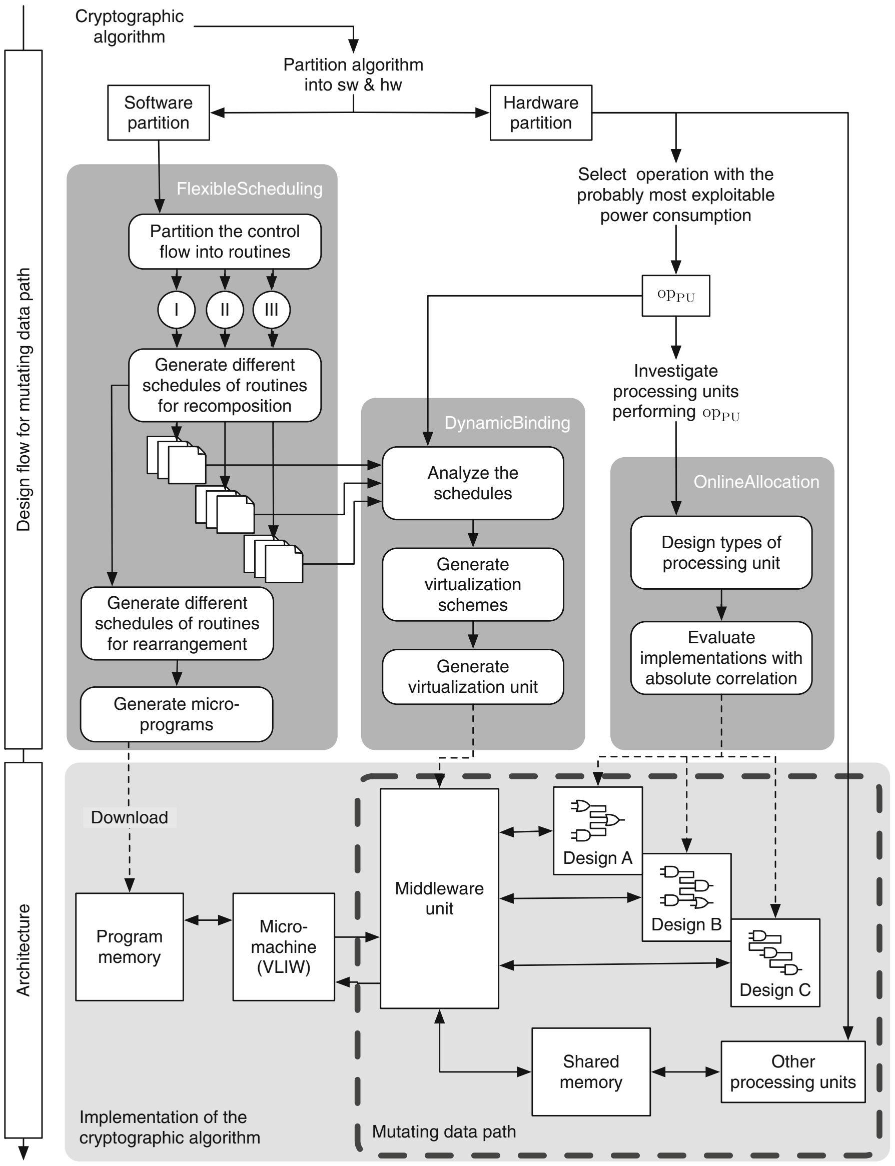

Cryptographic algorithm
Partition algorithm into sw & hw
Software partition
Hardware partition
Select operation with the probably most exploitable power consumption
op_PU
Investigate processing units performing op_PU
FlexibleScheduling
Partition the control flow into routines
I
II
III
Generate different schedules of routines for recomposition
Generate different schedules of routines for rearrangement
Generate micro-programs
DynamicBinding
Analyze the schedules
Generate virtualization schemes
Generate virtualization unit
OnlineAllocation
Design types of processing unit
Evaluate implementations with absolute correlation
Design flow for mutating data path
Architecture
Download
Program memory
Micro-machine (VLIW)
Middleware unit
Design A
Design B
Design C
Shared memory
Other processing units
Implementation of the cryptographic algorithm
Mutating data path

**Fig. 8.5** Design flow for and outline of the Mutating Runtime Architecture

## 8.4 Case Study: Block Cipher AES 128-Bit

In this section we present the application of the proposed concept on the well-known symmetric-key AES algorithm. The hardened design will from now on be referred to as AES Mutate. One property of the advocated MRA is the manipulation of in-parallel processed data, therefore the envisaged architecture of AES Mutate shall support a round-based as well as a non-round-based implementation style. The four basic operations of AES need to be modularized into efficient hardware components in order to support a 128-bit as well as some smaller word width processing. Therefore, the key scheduler has to be able to produce a complete new round key every clock cycle. An additional port is needed to attach a source of entropy in order to enable to mutate the block cipher non-deterministically.

### *8.4.1 Partitioning of the AES Modules*

The generic MRA outlined in the lower part of Fig. 8.5 needs now to be adapted and refined according to the requirements of AES Mutate. An overview of the HW/SW partitioning of AES Mutate and the role of construction methods are illustrated in Fig. 8.6. As visible from this figure the software partition located on the left-hand side controls both the data flow within and the configuration of the cipher. Therefore, from a controlling point of view, the behavior in terms of reconfiguring the system and executing an encryption task is easy to adapt. The VLIW opcode for the configuration of the data path as well as the opcode to control the basic operations for an encryption are implemented in hardware and are stored in the program memory. A RAM module of the FPGA platform can be utilized for this purpose. Then, a tiny micro-machine addresses this memory in the scheduled order. The functionality of the middleware is separated into a software part, embedded into the micro-machine, and into a hardware part. The software part of the middleware manages the control flow and the execution order in case of changing the binding. The hardware part consists of the reconfigurable data path, whereas the basic

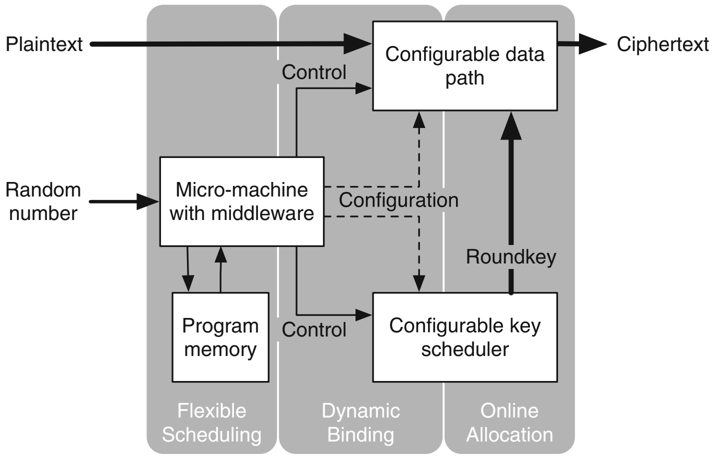


**Fig. 8.6** Countermeasure construction methods and HW/SW partitions of AES Mutate

operations for executing a round operation are embodied as hardware components. The required operations for expanding the secret key according to the round key are grouped into the hardware partition too. The hardware partition of the key scheduler located on the lower right-hand side of Fig. 8.6 is reconfigurable too, so that the variation of the round execution in terms of processing intermediate values in parallel or of execution length, respectively, does not affect the computational correctness of the result.

The design-specific reasons for partitioning the functionalities of the mutating procedure in this way are directly visible from Fig. 8.6. The interface between the construction methods FlexibleScheduling and OnlineAllocation provides the DynamicBinding functionality. This functionality in turn is realized jointly by components of the micro-machine and by architecture entities within both the data path and the key scheduler. More specifically speaking, middleware components are used to put in place DynamicBinding as the central modification instance according to the design flow of Fig. 8.5.

## 8.4.2 *Implementation*

In the sequel the implementation procedure of the block cipher is being detailed. First, the attack threat on AES is discussed in order to identify, which basic operation is most vulnerable due to its power consumption signature and thus has to be secured. Based on this discussion the different properties of the mutating architecture are reasoned in order to identify the parts of the data path, which have to be reconfigurable to increase the effort of a power analysis attack. The development of the cipher follows the generic design flow and exercises all proposed construction methods in order to harden the design by multiple countermeasures.

### 8.4.2.1 Design Requirements

The mutating architecture of the block cipher shall fulfill the following requirements and features:

- Executing of the SBox operation on quite different implementations
- Dynamically changing the degree of in-parallel processing SBox units
- Merging round operations into one clock cycle
- Manipulating the word width being processed during a clock cycle
- Randomizing the state representation in the shared memory.

The SBox operation is selected as the target of OnlineAllocation as well as for DynamicBinding, because it is the most vulnerable operation in two of three round operations. In the studied attack scenarios in [19], the SBox operation was the central element to determine the estimated exploitable power consumption of the whole block cipher. It is a bijective operation with a high amount of diffusion and, as reported in [15], it has the highest power consumption of all the basic operations of the AES. There is a wealth of various kinds of implementations of SBox units reported in literature featuring different design-specific characteristics, cf. [15, 18], which may be exploited to manipulate the power consumption of the operation.

Due to the higher power consumption of in-parallel processing SBox variants the variance of the power signature is enlarged. Thus, the SNR of extractable information within the captured power traces will decrease and thereby increase the attacking effort. Varying the number of operations per clock cycle and merging round operations (routines of the AES, cf. Fig. 8.4) are means to hide the amount of concurrently active processing SBoxes. In this context FlexibleScheduling may be seen as a combination of Shuffling [13] and Temporal Jitter countermeasures [14]. The last point in the list above is intended to prevent an exploitation of the XOR operation by kind of a randomized Register Pre-charging and Shuffling.

#### 8.4.2.2 Data Path Architecture

Figure 8.7 provides an overview on the structure of the data path. As mentioned before the data path for the transformation of the plaintext to the ciphertext (upper circuit part) and the data path for the key expansion (lower circuit part) have to be mutable. The required degree of in-parallel data processed per clock cycle is scalable in byte-wise steps from 8 bit up to 128 bit. The key scheduler circuit is extended by a multiplexor in order to switch between either the updated bytes of the round key value or the previous bytes of the round key. Thus, the key scheduler can process in-parallel the next round key on 128 bit at once or in 96, 64, or 32 bit wide chunks.

We selected a set of AES SBox implementations with different physical behavior and power signatures for usage in the P-units of Fig. 8.7. For the choice of SBox types we followed the recommendation in [19] due to the outlined evaluation and a mature selection process. In order to save space on the FPGA platform, a concurrent SBox management is in place to exchange the SBox type of each byte section of the AES: The virtualization method discussed above is used to change the binding between the state register and the SBox implementations in addition to partial reconfiguration. So, not every SBox in the data path needs to be partially reconfigurable as depicted in Fig. 8.5. This design decision helps to save resources and execution time.

The challenge of changing the degree of in-parallel processed data through the SBox unit stems from the entangled data dependency between columns and rows of the AES round state matrix. The SBox is embedded in the so-called P-unit, which also offers the functionality of the AES specific basic operations MixColumn and AddRoundkey as well as a bypass operation. The P-units work on the columns of the matrix and they are interconnected by a reconfigurable ShiftRows (RSR) function in order to process the complete state matrix. The RSR can shift each byte of a row from one byte to the right up to two bytes on the left as well as bypass the shift operation in order to maintain the word position of the four bytes in the rows. This unit can also be used to either shuffle the state representation on-the-fly

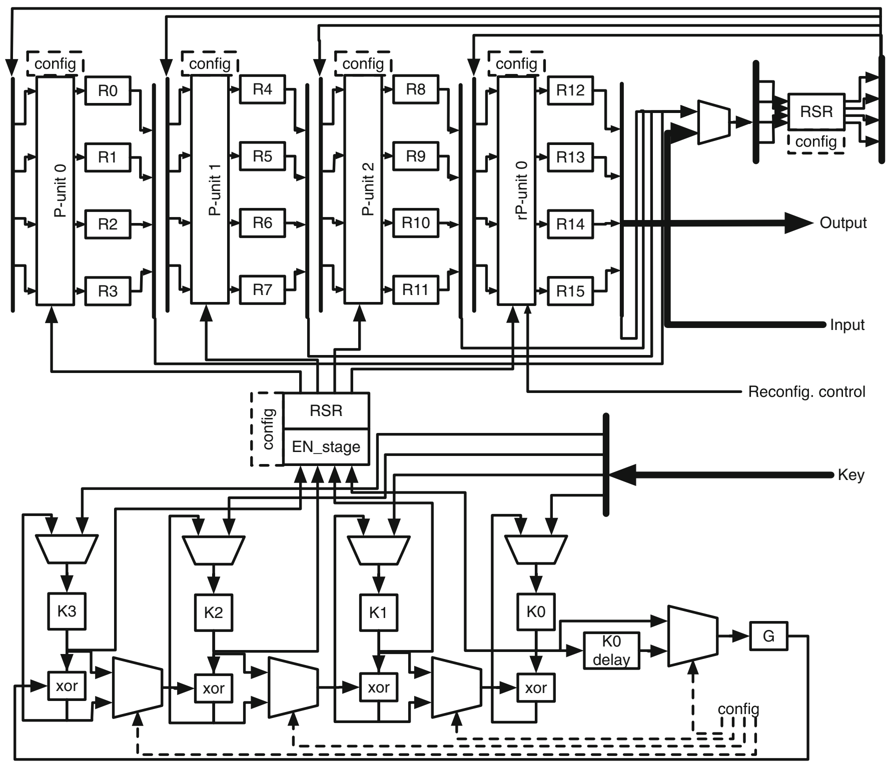


**Fig. 8.7** Schematic of the AES Mutate data path

or to change the binding to a specific SBox instance. In case of shuffling the state, the subkeys have also to be shuffled in the same order. Thus, an additional RSR unit is located between the P-units and the key scheduler. Therefore, the state randomization can be inserted within a round operation without needing any additional clock cycles for the reconstruction of the original value of the state register position.

The AddRoundkey function is merged with the MixColumn function to the so-called *MXOR*, which is configurable so as to perform the MixColumn operation followed by an XOR operation for the AddRoundkey functionality or just to execute an XOR. Thus, the P-unit provides three functionalities, which can either be performed in a combination at any level or may be executed individually,[1] and a bypass functionality. In order to support the partial reconfiguration on the Virtex 5 platform at moderate costs in terms of resources and execution time, the so-called reconfigurable P-unit (*rP-unit*) is added to this data path instead of a forth P-unit,

[1]SBox, MixColumn, AddRoundkey.

cf. Fig. 8.7. The rP-unit features an additional layer of multiplexors aimed to route data between the partial reconfigurable areas of the SBox. While one reconfigurable area is being updated with a new configuration, the other one continues to execute the SBox operation.

#### 8.4.2.3 Virtualization Scheme

As mentioned in Sect. 8.4.1, the data flow routing functionality of the middleware is embedded in the data path. A virtualization of this design can be conducted by utilizing the RSR units to reroute or to shuffle the content of the register states. Thereby, different SBox implementations may be utilized for each byte-width entry of the AES state matrix in case each P-unit/rP-unit instance features SBox implementation variants. The configuration settings of the P-unit/rP-unit are the second control element supporting the virtualization in order to realize the method DynamicBinding. Due to these settings the amount of in-parallel working SBoxes per clock cycle can be controlled.

As visible from Fig. 8.6 a micro-machine is part of AES Mutate, which executes a tailored VLIW opcode command set for controlling the configuration settings of the P-units/rP-unit as well as of the RSR units. Due to a flexible VLIW programming environment and the associated hardware architecture of the middleware this functionality is embedded into the data path. The outlined virtualization provides a suitable interface between the DynamicBinding and FlexibleScheduling methods. The configuration settings for a P-unit/rP-unit can be used in addition to modify the amount of in-parallel working hardware components. For instance, the execution of multiple XOR operations can be manipulated in terms of the degree of in-parallel working XORs as well as the amount of basic AES operations executed within one clock cycle.

#### 8.4.2.4 Merging Round and Routines

The modularized, flexible, and reconfigurable HW/SW architecture of AES Mutate is able to randomize the execution of every basic AES operation, because an application of FlexibleScheduling offers the ability to modify the degree of in-parallel working processing units of all basic AES operations. The width of the data path is adaptable for all operations resulting in 32, 64, 96, and 128 bit, respectively, processed within one clock cycle. The layout of the micro-machine in combination with the embedded middleware infrastructure present in the data path provides a flexible rescheduling feature via VLIW operations, which are independent of the current data path processing width setting. Therefore, the number of data-dependent operations processed within one clock cycle is randomizable and, thus, basic operations of different AES round operations can be merged.

The rescheduling of the AES encryption results from utilizing small micro-programs, which contain just one or few 36 bit wide VLIW opcode commands. Each round or merged round operation is represented by such a micro-program.

Within one clock cycle the opcode controls the processing units of the basic AES operations and sets the configuration of the data path. Thus, the variable word width of the data path is defined by just an opcode command.

The MSB bit of the opcode is used as a delimiter in order to identify the end of a round operation. Therefore, a complete AES encryption can be composed from randomly selecting different micro-programs for each round operation of the AES algorithm. The lower 32 bits of the opcode are used to reconfigure the data path during runtime in order to randomize the overall power consumption. These micro-programs are therefore more or less reconfiguration settings aimed to change dynamically the architecture of the data path. This context-based reconfiguration approach was chosen because the partial reconfiguration scheme offered by the Xilinx, Inc. design flow takes from our perspective too much time to complete. Therefore, we decided to aim for a reasonable tradeoff by using only one rP-unit next to three P-units and a modular micro-programming approach instead.

### *8.4.3 Side-Channel Analysis Results*

We compare the results gained from the hardened cipher to those of an unprotected implementation in order to emphasize the additional effort of attacking the secured design by means of a passive power analysis. The unprotected implementation is a round-based operating AES scheme implementation applying the COMP SBox variant and is denoted as AES COMP. The resulting block cipher processes 128 bits per clock cycle. The evaluation was conducted with 500,000 power traces in total, which were captured from the SASEBO-GII board. In case of the analysis of the AES Mutate design an additional trivium stream cipher [16] is introduced aimed to produce the random input data.

For a comparison of the side-channel resistance properties of AES Mutate and AES COMP the leakage of the SBox computation of the first round was analyzed. As analysis method we selected the Stochastic Approach [8], because it is a powerful profiling method that exploits the leakage at best by means of a self-learning linear regression model. We analyzed 450,000 power traces with random plaintext in the profiling phase to build the linear model. Then we captured 50,000 additional traces from the same device for the attack phase. Using the same device for both the profiling and the attack phase of a template-based analysis represents the optimal case for an attacker and, hence, should yield as the evaluation result the lower bound of security in terms of side-channel resistance. As distinguisher and evaluation metric for detecting exploitable leakage we applied the SNR-metric introduced in [19].

Figure 8.8 depicts the results of the side-channel analysis of these implementation variants on top of the Xilinx Virtex 5 platform. The figure visualizes the amount of correctly revealed secret key bytes over the number of required traces during the attack phase. One can easily derive from this figure that the first correct subkeys extracted from the unprotected AES COMP design were unveiled after just a few hundred traces and all secret subkeys were recovered after using 17,780 traces in total. The attack result of the AES Mutate design hardened by multiple

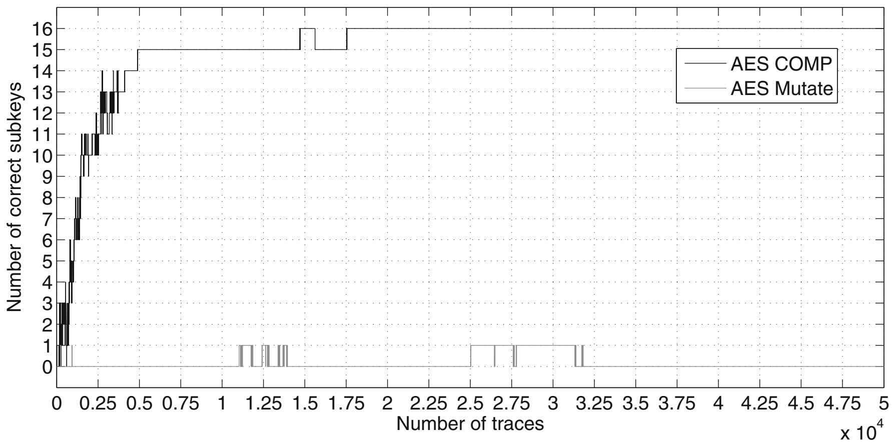


**Fig. 8.8** Comparison of the analysis results of AES COMP and AES Mutate ciphers

countermeasures shows in contrast an unstable extraction of just one out of 16 subkeys. Hence, the secret key keeps its strength during the attack phase for the complete set of applied traces. The unstable analysis result nicely indicates that the advocated concept of mutating cipher architectures works very well in order to generate a wide noise distribution intended to protect the real key-characteristic distribution.

## 8.5 Summary

We presented in this chapter a novel technique to secure cryptographic modules of embedded systems against power analysis attacks, which form the most popular class of passive, non-invasive SCA strategies. Taking the inherent reconfiguration properties of FPGAs and the generic design methodology of computing systems as foundations, we first introduced construction methods for multiple countermeasures and presented subsequently a Mutating Runtime Architecture, which is well-suited to harden quite different cipher types. We then detailed as proof of concept a case study aimed to secure an FPGA implementation of the AES algorithm and highlighted the advantage of an automatic generation of multiple countermeasures.

**Acknowledgements** This work was funded in part by DFG, the German science foundation, under grant no. HU620/12 within the DFG Priority Programme 1148 “Reconfigurable Computing Systems” in cooperation with CASED.

## References

1. L. Benini, A. Macii, E. Macii, E. Omerbegovic, M. Poncino, F. Pro, A novel architecture for power maskable arithmetic units, in *GLSVLSI* (ACM, New York, 2003), pp. 136–140
2. L. Benini, A. Macii, E. Macii, E. Omerbegovic, F. Pro, M. Poncino, Energy-aware design techniques for differential power analysis protection, in *DAC* (ACM, New York, 2003), pp. 36–41
3. D. Canright, A very compact Rijndael S-Box. Technical Report, Naval Postgraduate School (2005)
4. M. Ernst, M. Jung, F. Madlener, S.A. Huss, R. Blümel, A reconfigurable system on chip implementation for elliptic curve cryptography over GF($2^n$), in *CHES*. Lecture Notes in Computer Science, vol. 2523 (Springer, Berlin, 2002), pp. 381–399
5. W. Fischer, B.M. Gammel, Masking at gate level in the presence of glitches. in *CHES*, ed. by J.R. Rao, B. Sunar. Lecture Notes in Computer Science, vol. 3659 (Springer, Berlin, 2005), pp. 187–200
6. D.D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner, *Embedded System Design: Modeling, Synthesis and Verification*, 1st edn. (Springer, Berlin, 2009)
7. B. Jungk, M. Stöttinger, J. Gampe, S. Reith, S.A. Huss, Side-channel resistant AES architecture utilizing randomized composite-field representations, in *FPT* (IEEE, New York, 2012), pp. 125–128
8. M. Kasper, W. Schindler, M. Stöttinger, A stochastic method for security evaluation of cryptographic FPGA implementations, in *FPT* ed. by J. Bian, Q. Zhou, P. Athanas, Y. Ha, K. Zhao (IEEE, New York, 2010), pp. 146–153
9. P.C. Kocher, J. Jaffe, B. Jun, Differential power analysis, in *CRYPTO 99*, ed. by M.J. Wiener. Lecture Notes in Computer Science, vol. 1666 (Springer, Berlin, 1999), pp. 388–397
10. F. Madlener, M. Stöttinger, S.A. Huss, Novel hardening techniques against differential power analysis for multiplication in GF($2^n$), in *FPT* (IEEE, New York, 2009)
11. S. Mangard, T. Popp, B.M. Gammel, Side-channel leakage of masked CMOS gates, in *CT-RSA*, ed. by A. Menezes. Lecture Notes in Computer Science, vol. 3376 (Springer, Berlin, 2005), pp. 351–365
12. S. Mangard, N. Pramstaller, E. Oswald, Successfully attacking masked AES hardware implementations, in *CHES*, ed. by J.R. Rao, B. Sunar. Lecture Notes in Computer Science, vol. 3659 (Springer, Berlin, 2005), pp. 157–171
13. S. Mangard, T. Popp, M.E. Oswald, *Power Analysis Attacks - Revealing the Secrets of Smart Cards* (Springer, Berlin, 2007)
14. N. Mentens, B. Gierlichs, I. Verbauwhede, Power and fault analysis resistance in hardware through dynamic reconfiguration, in *CHES*, ed. by E. Oswald, P. Rohatgi. Lecture Notes in Computer Science, vol. 5154 (Springer, Berlin, 2008), pp. 346–362
15. S. Morioka, A. Satoh, An optimized S-box circuit architecture for low power AES design, in *CHES*, ed. by B.S.K. Çetin Kaya Koç Jr., C. Paar. Lecture Notes in Computer Science, vol. 2523 (2002), pp. 172–186
16. C. Paar, J. Pelzl, *Understanding Cryptography - A Textbook for Students and Practitioners* (Springer, Berlin, 2010)
17. F. Regazzoni, W. Yi, F.X. Standaert, FPGA implementations of the AES masked against power analysis attacks, in *COSADE* (2011), pp. 55–66
18. A. Satoh, S. Morioka, K. Takano, S. Munetoh, A compact Rijndael hardware architecture with S-Box optimization, in *ASIACRYPT*, ed. by C. Boyd. Lecture Notes in Computer Science, vol. 2248 (Springer, Berlin, 2001), pp. 239–254
19. M. Stöttinger, Mutating runtime architectures as a countermeasure against power analysis attacks. PhD thesis, Technische Universität Darmstadt (2012)

20. M. Stöttinger, A. Biedermann, S.A. Huss, Virtualization within a parallel array of homogeneous processing units, in *ARC*, ed. by P. Sirisuk, F. Morgan, T.A. El-Ghazawi, H. Amano. Lecture Notes in Computer Science, vol. 5992 (Springer, Berlin, 2010), pp. 17–28
21. T. Sugawara, N. Homma, T. Aoki, A. Satoh, Differential power analysis of AES ASIC implementations with various S-box circuits, in *ECCTD* (IEEE, New York, 2009), pp. 395–398