Side Channel Attack Counter Measure using a Moving Target Architecture

Jithin Joseph
University of New Mexico - Main Campus

Follow this and additional works at: https://digitalrepository.unm.edu/ece_etds

Part of the Digital Circuits Commons, Electrical and Electronics Commons, Hardware Systems Commons, Other Computer Engineering Commons, and the VLSI and Circuits, Embedded and Hardware Systems Commons

Recommended Citation

This Thesis is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Electrical and Computer Engineering ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.
Jithin Joseph

Candidate

Electrical and Computer Engineering

Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Dr. James Plusquellic

Committee Chair

Dr. Payman Zarkesh-Ha

Committee Member

Dr. Eirini Eleni Tsiropoulou

Committee Member
Side Channel Attack Counter Measure using a Moving Target Architecture

by

Jithin Joseph

B.Tech., Rajagiri School of Engineering and Technology, 2015

THESIS

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science
Computer Engineering

The University of New Mexico
Albuquerque, New Mexico

May, 2021
Acknowledgments

I would like to thank Dr. James Plusquellec, for his guidance and patience. I would also like to thank my family for their constant support, even though they were an ocean away.
Side Channel Attack Counter Measure
using a Moving Target Architecture

by

Jithin Joseph

B.Tech., Rajagiri School of Engineering and Technology, 2015
M.S., Computer Engineering, University of New Mexico, 2021

Abstract

A novel countermeasure to side-channel power analysis attacks called Side-channel Power analysis Resistance for Encryption Algorithms using DPR or SPREAD is investigated in this thesis. The countermeasure leverages a strategy that is best characterized as a moving target architecture. Modern field programmable gate arrays (FPGA) architectures provide support for dynamic partial reconfiguration (DPR), a feature that allows real-time reconfiguration of the programmable logic (PL). The moving target architecture proposed in this work leverages DPR to implement a power analysis countermeasure to side-channel attacks, the most common of which are referred to as differential power analysis (DPA) and correlation power analysis (CPA). The goal of the SPREAD technique is to reduce the power transient signal correlations associated with repeated use of the cryptographic key during encryption operations by dynamically changing the logic level hardware implementation of the cryptographic engine in real time.
# Table of Contents

List of Figures vii  
List of Tables ix  

1 Introduction 1  

2 Literature Survey 4  

3 Architecture Design 7  

3.1 Block Diagram 9  
3.2 Implementation Diversity Model 11  
3.3 Synthesis-Directed Implementation Diversity 12  
3.4 Diversity Through Clusters 14  
3.5 Relocatable Partial Bitstreams 16  

4 Experiments 20  

4.1 Experimental Setup 20
Table of Contents

4.1.1 Attack Model .................................................. 23

4.2 Experimental Results ........................................... 25

4.2.1 SPREAD Versions .............................................. 25

4.2.2 CPA Evolutionary Graphs .................................... 27

5 Conclusion ....................................................... 31

References .......................................................... 32
List of Figures

3.1 Block diagram of the MUX structure that controls the selection of the SBOXs. The control signals needed for this task are generated by the master controller. SBOX\textsubscript{14} and SBOX\textsubscript{0} are disconnected from the datapath in this configuration. ............................................. 8

3.2 Block diagram of the proposed architecture, consisting of the AES engine, the internal configuration access port (ICAP), the block memory (BRAM) that holds the 18 partial bitstreams and the master controller that also contains the nonce generator. .................. 10

3.3 Synthesis directed CAD tool flow. ................................................... 13

3.4 A comparison between the implementation of two SBOX designs generated using the synthesis directed implementation diversity approach. The image highlights how the structural diversity translates to implementation diversity. .................................................. 14

3.5 Delay diversity obtained by the synthesis directed method for a subset of paths in 8 SBOX designs. .................................................. 16

3.6 Implementation view of the static design produced by the IMPRESS tool flow that illustrate the 18 relocatable reconfiguration regions for SBOX components of the SPREAD engine. .......................... 18
List of Figures

4.1 Block diagram of the experimental setup. . . . . . . . . . . . . . . . 21
4.2 SAKURA-X board used in the experiments and example power trace from round 1 of the AES encryption engine. . . . . . . . . . . . . . . 22
4.3 Block diagram of the AES engine showing the datapath. . . . . . . 26
4.4 Circuit used to introduce delay (jitter) to the clock inputs of the dat registers in the datapath component of the AES engine. . . . . . . . 27
4.5 Evolutionary plot associated with Byte 7 of the reference design for version V6. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28
4.6 Evolutionary plot associated with Byte 7 of the countermeasure for version V6. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
4.7 Comparison of average reduction in signal correlation for all 4 versions of the proposed SPREAD design relative to the levels of signal correlation present in the reference designs of each version. . . . . . 30
List of Tables

3.1 The different synthesized designs of the SBOX utilize different combinations of logic gates. This demonstrates the effectiveness of the synthesis directed approach in generating implementation diversity. 13
Chapter 1

Introduction

Recent years have seen the concepts of security and trust becoming increasingly important in emerging and existing electronic systems including those in industry, defense, aerospace, data centers, supervisory and data acquisition (SCADA) environments and health care devices [1, 2, 3]. These systems are becoming increasingly vulnerable as they are left unsupervised in the field with internet connectivity.

Data integrity and privacy of communication between devices is ensured by relying heavily on authentication and encryption techniques. The root of trust in these systems are the cryptographic keys and so it is important that their secrecy is ensured at all times including when the system is powered off. Secrecy is traditionally ensured mathematically by making the process of reverse engineering chips computationally complex. The attacker is assumed to have control only over the digital inputs and is only able to observe the encrypted outputs. Physical hardware unfortunately has several unintended I/O channels called side-channels through which an attacker can extract additional information about the cryptographic key. The attacker can use this side-channel information to significantly reduce the time and effort required to reverse engineer the cryptographic key [4, 5]. Device parameters that can be
leveraged as side-channels include leakage current, dynamic power (transient currents) and electromagnetic emissions. The adversary can use these techniques along with fault injection techniques, where they intentionally introduce clock and power glitches [6, 7], to reveal secret keys and other sensitive information in a matter of hours or days.

The goal of this research is to investigate a countermeasure to these side-channel attacks (SCA). Our focus is to develop a method that can reduce the effectiveness of differential (DPA) [5] and correlation power analysis (CPA) [8], but it might also be effective against electromagnetic analysis (EMA) [9, 10, 11]. Power analysis techniques (1) provide visibility into the gate-level switching behavior of the chip; (2) are non-destructive and semi-invasive and do not need expensive equipment to utilize; (3) have been successful even in the presence of countermeasures [12, 13], hence they are particularly problematic.

In this research we utilize the dynamic partial reconfiguration (DPR) feature available in modern field programmable gate arrays (FPGAs) hardware. The proposed architecture deliberately changes the physical layer implementation characteristics of an encryption algorithm in real time using DPR. By dynamically changing the hardware configuration, a key assumption that makes power analysis attacks so powerful no longer holds true.

The proposed moving target architecture self-reconfigures a part of the circuit implementation, in case of the advanced encryption standard (AES), the SBOX, while simultaneously enabling encryption or decryption operations to proceed in parallel. This is accomplished by introducing an extra copy(s) of the SBOX into the architectural design. The SPREAD engine applies control signals to a set of MUXes that re-wire the data path of the AES engine to remove the redundant SBOX and then reprograms the redundant SBOX with a functionally equivalent partial bitstream that posses a diverse logic and routing structure. The SBOX is
Chapter 1. Introduction

then ‘re-installed’ into the AES datapath and another SBOX is randomly selected as the redundant copy for reprogramming. The SPREAD engine is implemented as a state machine running in parallel with AES, and therefore this strategy is referred to as self-reconfiguration. Cryptographic operations of the encryption engine run at full speed and in parallel with the reprogramming functionality with at most only a one clock cycle stall to reconfigure the MUX network.

This moving target architecture can be considered a hybrid of previously proposed noise enhancing and signal reducing countermeasures because SPREAD introduces noise into the power traces that is best described as uncorrelated signal information. In other words, signal power is introduced as the source of noise because the architecture changes parts of the hardware implementation on the fly.

A synthesis-directed technique is described to introduce implementation diversity into the SBOX functionality. The proposed architecture is then evaluated using the correlation power analysis (CPA) attack algorithm, which in our experience is the most effective of the proposed SCA attack strategies. The power trace data is collected using an FPGA platform optimized specifically for enhancing the effectiveness of side-channel attacks.
Chapter 2

Literature Survey

Countermeasures can be classified either into algorithmic countermeasures that mask security critical processes of cryptographic operators or hardware countermeasures that add noise via non-deterministic operations or leverage side-channel resistant logic styles. Their effectiveness is measured by counting the number of plaintext encryptions required for the secret key to be successfully extracted.

Reconfiguration has been proposed as a countermeasure previously and can be classified into two types, dynamic partial reconfiguration (DPR) and dynamic logic reconfiguration (DLR). Both types change the underlying hardware on the fly. DLR selects between multiple copies of a functional unit that has been preprogrammed into the FPGA fabric while DPR programs the FPGA with unique functional units that have different logic structures on the fly. A DLR technique was proposed by Sasdrich et al [15] for the PRESENT cipher which combines several copies of the Xilinx Configurable Lookup Tables (LUTs) to create reconfigurable function tables. The authors of [16] propose a scheme that utilizes masked LUTs to implement block memory content scrambling. This technique makes use of the Xilinx SLICE-M LUTs for building randomly permuted SBOXs. The authors of [17] succeeded in creating
confusion and thereby increase side-channel attack resistance on resource constrained
implementations by utilizing randomized isomorphisms of the algebraic construction
of SBOXs.

A countermeasure against leakage attacks utilizing DPR was suggested by
Mertens et al. [18] which randomly adds registers between functional units of the
design using a dynamically reconfigurable switch matrix. This feature adds tempo-
ral jitter. They also randomly move the functional blocks around which adds spatial
jitter. The authors of [19] propose an authentication technique to use for the transfer
of partial bitstreams used in DPR operations. Huss et al. [20] combines three sepa-
rate countermeasures to create a self-developed mutating runtime architecture which
aims to create confusion and diffusion in the standard AES engine implementation
running on an FPGA.

A generic, resource efficient countermeasure was proposed by Güneysu et al. [21]
for random data processing, on-chip noise generation and SBOX scrambling by mak-
ing use of dual ported block memory. Swankiski et al. [22] proposes the use of
a parallel architecture in order to isolate the key in temporal space. Countermea-
sures to hide leakage information were proposed by Shan et al. [23] by using idle
reconfigurable processing elements to perform dummy operations. Levi et al. [24]
proposes a technique that assigns buffer delays to combinational logic using a delay
assignment algorithm to implement data dependant delays. This is done to combat
single and multi bit CPA attacks. They also describe in [25] a pseudo-asynchronous
design methodology which utilizes both synchronous and asynchronous designs with
the goal of making the alignment of power traces to extract secret information diffi-
cult. This design technique is combined with randomization and data dependencies
to prevent information leakage.

The authors of [26] proposes a method that uses DPR where the entire AES core
is reprogrammed with a different implementation over a period of time. A significant
performance penalty will be paid with this technique because the encryption operations need to be halted during reconfiguration. This technique also requires partial bitstreams of considerable size totaling in excess of 700 MB of external storage.

The moving target architecture proposed in this research is different from previous work in several ways. Our proposed countermeasure introduces a larger amount of entropy than other techniques since it uses a logic level diversity technique and does not just rearrange existing functional blocks of the implementation. Our technique also utilizes minimal chip area since it leverages dynamic partial relocation and relocatable bitstreams which allows for the same partial bitstream to be used to program different locations on the FPGA by modifying its frame address. Finally, the proposed architecture introduces temporal distortion by moving functional units around randomly and dynamically. This countermeasure is based on ideas explored in [27, 28] but is different from the concept described in [20].
Chapter 3

Architecture Design

The countermeasure proposed in this research uses DPR and is best characterized as a moving target architecture. A block diagram of the proposed architecture is shown in Fig 3.1, where we show only the SBOX portion of the AES engine. For a 128-bit implementation of AES, we require 16 SBOXs in the datapath arranged in parallel as shown. Any of these SBOXs can be dynamically reprogrammed on the fly by the DPR engine. To accomplish this, we add two additional redundant SBOXs in parallel with the existing 16. These redundant SBOXs can be shifted to the left or right using a MUX’ing structure to shuffle data processing to neighboring SBOX locations, or they can also serve as the target for a reprogramming operation. The location of the redundant SBOXs is determined by the DPR engine which generates DPR control signals. These signals drive the control signals of the top and bottom row of selection MUXs, which modifies the datapath accordingly. Fig 3.1 shows a configuration where SBOX\textsubscript{14} and SBOX\textsubscript{0} have been MUX’ed out of the datapath.

The time required to reconfigure one of the SBOX regions is approximately 100\(\mu\)s. If encryption could not proceed in parallel with reconfiguration, the proposed approach would not be practical because of the significant reduction in throughput.
Chapter 3. Architecture Design

Figure 3.1: Block diagram of the MUX structure that controls the selection of the SBOXs. The control signals needed for this task are generated by the master controller. SBOX_{14} and SBOX_{0} are disconnected from the datapath in this configuration.

that would result. Fortunately, in most FPGA architectures, such as Xilinx, the reconfiguration operation can proceed independent of the rest of the system. Therefore, our proposed architecture only introduces a single stall cycle to reconfigure the MUX’ing structure.

It is also important to note that we follow a non-standard Vivado tool flow to construct SBOX partial bitstreams that are relocatable because Xilinx does not support relocatable partial bitstreams. The relocatable characteristic allows any of the SBOX partial bitstreams to be used to reprogram any of the 18 SBOX regions, i.e., only one copy of each diverse implementation is required. Regions of the FPGA that are designed to support relocation require static logic and routing belonging to the top level design to be excluded, i.e., blocked, from these reconfigurable regions. The standard DPR flow supported within Xilinx’s Vivado CAD tool considers regions
designated as reconfiguration regions as legitimate targets for the placement and routing of components in the static design. Therefore, special processing is required to prevent Vivado from utilizing these regions. Fortunately, the community has developed a tcl-based synthesis tool called IMPRESS [33] that makes it impossible for Vivado to route into regions designated as reconfigurable and relocatable. The IMPRESS tool made it possible to reduce the number of partial bitstream to 18 (one for each region) and to store the entire set of partial bitstreams in block memory (BRAM) resources available on the FPGA. This enabled reconfiguration to proceed at high speed using only on-chip resources.

The remaining functional components of the AES engine, including Mix-Columns, Shift-Rows and Add-Round, are connected combinationally to the outputs of the SBOXs and are not included in the reconfigurable regions.

3.1 Block Diagram

Fig 3.2 depicts a block diagram of the proposed architecture. The system consists of the AES engine with 18 SBOXs MUXed into the datapath as shown in Fig 3.1, the internal configuration access port (ICAP), the on-chip block memory (BRAM) which stores the 18 partial bitstreams and a master controller block which generates the necessary control signals. The ICAP interface enables self-reconfiguration of the FPGA. A partial bitstream can be streamed into this interface along with the required control signals and the FPGA reprograms the designated region. We incorporated the partial bitstreams into the full bitstream by adding a BRAM memory in Vivado, and then specifying the initial contents using a coefficient file (COE). Therefore, after programming the FPGA, the SPREAD engine is fully instantiated and ready for operation.

An overview of system functionality is given as follows: the DPR controller starts
Chapter 3. Architecture Design

the internal nonce generator and uses the nonce to randomly select the partial bitstream, the reconfiguration region and the time between reconfigurations. The DPR controller then synchronizes with the AES engine and generates the required control signals to configure the MUXing structure and to transfer a partial bitstream over the Xilinx ICAP interface.

Figure 3.2: Block diagram of the proposed architecture, consisting of the AES engine, the internal configuration access port (ICAP), the block memory (BRAM) that holds the 18 partial bitstreams and the master controller that also contains the nonce generator.

The time taken to complete a DPR operation on an SBOX region is approximately 100µs and the power consumption is approximately 20 µW. The rate of reconfiguration is bounded by the energy consumption overhead and the required SCA resistance. The achievable SCA resistance is related inversely with the number of power traces that can be collected under a particular configuration.
Chapter 3. Architecture Design

3.2 Implementation Diversity Model

The proposed moving target architecture aims to increase resistance to CPA and DPA attacks by reducing signal correlations in the power traces. The countermeasures introduced in this approach are neither the \textit{noise enhancing} type nor are they strictly the \textit{signal reducing} type. The architecture proposed is an inseparable combination of both types. The moving target architecture significantly reduces the correlations leveraged by SCA algorithms by making the signals themselves the source of noise.

These features challenge traditional uses of CPA and DPA. The adversary may for example require a large number of plaintexts to be applied in order to achieve the necessary signal averaging to reduce environmental and chip/board related noise, such as those generated due to temperature/voltage variations or trigger jitter. For this signal averaging process to be effective, the underlying hardware implementation needs to be unchanging over the power trace collection period. The reconfiguration operations carried out by the moving target architecture reduces the size of the time window over which signal averaging is possible. Therefore the collected power traces used for averaging will likely belong to different AES configuration states.

The number of possible AES configuration states is given by $n^k$, where $n$ represents the number of reconfigurable SBOX regions, and $k$ represents the number of possible implementations. The DPR controller is designed to make any of the $n^k$ states equally likely at any point in time. This large spatial diversity is combined with temporal diversity using a nonce driven controller algorithm. Finally, the architecture introduces spatial diversity even within a particular configuration state, that is, when no reconfiguration is performed to replace an SBOX implementation. This is achieved by configuring the MUX'ing structure in such a way that one of the redundant SBOX regions moves around randomly to different locations. This type of reconfiguration is expected to further reduce power trace correlations. The second
redundant SBOX region allows this type of shifting operation to occur at high frequency, e.g., between plaintext encryptions, and in parallel with the lower frequency DPR operations.

### 3.3 Synthesis-Directed Implementation Diversity

The goal of synthesis directed implementation diversity is to change the netlist and place and route characteristics of the SBOX. Our methodology makes use of different versions of a standard cell library. Standard cell libraries are used primarily in the design of application specific integrated circuits (ASICs). A register-transfer-level compiler, for example Cadence Genus, is used to convert a behavioral description of a digital circuit into a gate level structural netlist using the gates available in the standard cell library. We manipulate the gates available in this library to force the RTL compiler to synthesize a structural netlist using a different combination of logic gates. The new implementation of the structural netlists will have unique path delays and power trace behavior. The structural netlists are then used as input to the FPGA synthesis tool, Xilinx Vivado. Vivado performs further optimizations on the netlists before mapping them into FPGA fabric primitives, which act to further increase the diversity among the various implementations.

Fig 3.3 shows the computer aided design tool flow associated with synthesis-directed diversity. The behavioral description of the SBOX is converted into netlists using the RTL compiler tool, Cadence Genus. Each netlist is synthesized using a unique subset of logic gates present in the master standard cell library. These subsets are labelled StdCellLib\textsubscript{1}, StdCellLib\textsubscript{2} and so on. Table 3.1 demonstrates the implementation diversity achieved using this technique. The three SBOX designs shown utilize different combinations of gates to achieve the same functionality, but the structural differences will result in path delay variations and ultimately, in vari-
Chapter 3. Architecture Design

Figure 3.3: Synthesis directed CAD tool flow.

Fig 3.4 shows the results of performing implementation on the netlists generated using this technique. As can be seen from the figure, the two SBOX implementations have considerably different routing and placements. These layout images also serve to verify that the implementation diversity introduced by the standard cell flow does not get optimized back to the original, smaller SBOX implementation by Vivado during the synthesis and implementation phases.

<table>
<thead>
<tr>
<th></th>
<th>AND2X4</th>
<th>AND3X4</th>
<th>AND4X4</th>
<th>AND5X4</th>
<th>AND6X4</th>
<th>EXT10_d1_6X4</th>
<th>EXT10_d2_5X4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design1</td>
<td>24</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>17</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Design2</td>
<td>36</td>
<td>4</td>
<td>4</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>39</td>
</tr>
<tr>
<td>Design3</td>
<td>0</td>
<td>12</td>
<td>5</td>
<td>7</td>
<td>17</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.1: The different synthesized designs of the SBOX utilize different combinations of logic gates. This demonstrates the effectiveness of the synthesis directed approach in generating implementation diversity.
Chapter 3. Architecture Design

3.4 Diversity Through Clusters

We generated a set of 166 unique netlists using this process and selected the ’best’ 18 netlists for use as relocatable SBOX implementations for SPREAD using a clustering algorithm. In order to maximize the diversity in the 18 selected netlists, we made use of the K-means clustering algorithm to separate the netlists into 18 unique clusters. A clustering algorithm is designed to group similar sets of objects together and finds use in the fields of machine learning, pattern recognition and computer graphics, among others. The algorithm operates as follows. First, each netlist is considered to be a point in N-dimensional space where N is the number of unique logic gates present in the standard cell library used to synthesize the netlist. The coordinates of the netlist in this space are determined by the number of instances of each unique

Figure 3.4: A comparison between the implementation of two SBOX designs generated using the synthesis directed implementation diversity approach. The image highlights how the structural diversity translates to implementation diversity.
logic gate in the netlist. We first randomly select 18 netlists to act as initial cluster centroids. Each netlist is then assigned to a cluster by calculating the Euclidean distance of the netlist to the cluster centroids. The total Euclidean distance of the system is then evaluated by adding up the individual Euclidean distances between cluster centroids and cluster members. New cluster centroids are then calculated by finding the mean values of the coordinates of the cluster members. We then repeat this process as long as the total Euclidean distance of the system decreases with each iteration. After the system reaches equilibrium, we randomly select one netlist from each cluster. This process ensures that the final selected set maximizes the implementation diversity when using standard cell type and number as metrics.

The diversity of the path delays among implemented designs can be determined using timing-based simulation experiments. Since the relationship between path delays within the SBOXs and its corresponding power signal behavior is causal, we expect the analysis of delay to be predictive of the diversity which will occur in the power traces. Fig 3.5 illustrates a subset of the delay simulation results of the SBOX from post implementation timing simulations. Path delays are measured between the inputs and outputs of the SBOXs as two vector sequences are applied to the 8-bit SBOX inputs. The x-axis plots a delay number that corresponds to a path timed by the applied two vector sequence. Even though the SBOXs are functionally equivalent, the implementation diversity resulting from the unique structural definitions of the SBOX netlists makes the input-output delays distinct. Path delays vary from a few 100 ps to over 5 ns.

Fig 3.5 shows that the path delay for a particular SBOX instance in an invariant architecture will be represented by a single curve. However, the moving target architecture proposed in this research, which randomly reprograms the SBOXs with diverse implementations, causes the input-output path delays to change randomly and dynamically over time. Hence, for a given plaintext-key byte input, there are
Chapter 3. Architecture Design

Figure 3.5: Delay diversity obtained by the synthesis directed method for a subset of paths in 8 SBOX designs.

now 16 possible power trace patterns for each path through each SBOX.

3.5 Relocatable Partial Bitstreams

In the SPREAD moving target architecture, the partial bitstreams used to reconfigure the SBOX regions using DPR are designed to be relocatable. This means that any of the partial bitstreams can be used to reconfigure any of the reprogrammable SBOX regions. However, the default Vivado DPR tool flow does not support relocatable partial bitstreams, so another tool flow called IMPRESS is used. IMPRESS [33] is a set of tool command language (TCL) scripts that run within the Vivado TCL console. IMPRESS performs the necessary steps required to ensure relocatable partial bitstreams while using the Vivado synthesis and place&route tools to produce the bitstreams.
Chapter 3. Architecture Design

The primary issue that needs to be addressed regarding the standard Vivado tool flow is preventing Vivado from utilizing wiring and logic resources within each of the reconfiguration regions for components in the top-level static design. Vivado views these resources as legitimate targets because the resources used by earlier synthesis processes which create components for these regions are known by Vivado when the top level synthesis is performed. Given that each region has a separate set of partial bitstreams dedicated for that region, Vivado can create a database of the resources used across all partial bitstreams and decide which resources are available in all partial bitstreams. It then proceeds to place identically constructed static components in each copy of the partial bitstreams. The presence of the static top-level components 'locks' the partial bitstreams to this specific region in the FPGA fabric. A necessary condition to allow a region to be programmed with partial bitstreams synthesized for other regions of the design is the exclusion of all static top-level logic from the region. IMPRESS takes care of this issue by routing a temporary wire fence around the reconfigurable regions prior to asking Vivado to route the static design [33]. The fence utilizes all possible routes into the regions, effectively preventing Vivado from utilizing any unused resources within the region. Fig. 3.6 shows the implementation of the top level design of the SPREAD engine after running the IMPRESS tool flow. The 18 reconfigurable regions are highlighted as magenta rectangles, with 6 regions shown in three distinct rows. The reconfigurable regions are empty in this implementation view with only the positions of SBOX inputs and outputs annotated with white squares. Note that the all 18 regions must be placed over FPGA fabric regions that are structurally identical. This constraint imposes limits on the possible x coordinate locations for the regions. As a consequence, the 6 regions in each row are distributed non-uniformly across the row at locations where the underlying FPGA fabric is identical.
The IMPRESS tool flow also produces a set of 18 partial bitstreams, each associated with one of the diversified SBOX implementations described earlier. IMPRESS ensures that the SBOX inputs and outputs in all 18 bitstreams are located in precisely the same locations as a means of enabling any of the partial bitstreams to
connect to the existing static routes in the static top level design when programmed into a region. Given the FPGA fabric is identical under each of the regions as discussed above, an arbitrary SBOX partial bitstream can be ‘relocated’ to any region by simply changing the frame address (FAR) in the partial bitstream. In SPREAD, the 18 possible address locations are encoded as constants in the static design, and a state machine, the DPR controller, is designed that is capable of modifying the FAR at run time to any of these stored addresses as the partial bitstream is read from BRAM and streamed into the ICAP interface.

Although creating the relocatable bitstreams requires additional effort, the relocatable feature saves significantly on FPGA BRAM resources. For example, without relocatable partial bitstreams, the FPGA would need to store $18 \times 18 = 324$ versions of the SBOX implementations to enable the same capability as shown here with only 18 relocatable partial bitstreams. Moreover, given that each partial bitstream is approximately 25 KB in size, it would not be possible to create a programming bitstream with all 324 copies because of the limited on-chip BRAM resources, as we are able to do here with only 18. By embedding the relocatable partial bitstreams in the programming bitstream, we also increase the security of the system over alternative strategies which require external storage and run time access to a larger number of partial bitstreams.
Chapter 4

Experiments

4.1 Experimental Setup

The SAKURA-X FPGA board is used as the hardware platform for the side channel attack experiments [29]. The board includes a 1 Ohm resistor in series with the power supply pins of the Xilinx Kintex XC7K70T FPGA, which allows high quality power transient signals to be measured through a single ended or differential pair of board-mounted SMA connectors. The board also includes a Xilinx Spartan XC6SLX45 FPGA which acts as an interface between the Kintex and a peripheral computer through a USB bus. Fig 4.1 shows a block diagram of the test setup which includes the two FPGAs.

The SPREAD engine implementation investigated here is not fully implemented in the programmable logic of the FPGA. Instead, we implement a hybrid version that incorporates the most important elements of the proposed countermeasure in the FPGA (as described in the previous section) and implement the top level state machine components in a C program. The C program utilizes the USB bus interface to issue control signals to the FPGA state machines, and therefore, emulates the
Chapter 4. Experiments

Figure 4.1: Block diagram of the experimental setup.

full blown implementation. The emulated system provides a higher degree of control than would be possible in a pure PL side implementation of SPREAD. The focus of this research is first on determining the best set of countermeasures (discussed further below), which is facilitated by the hybrid implementation by enabling a wide range of options to be explored.

The C program provides complete control over the configuration process, the level of signal averaging, the selection of plaintexts to be encrypted, the frequency of reconfiguration and other timing characteristics associated with the data collection process. The C program controls the SPREAD engine through the USB interface using a set of commands that we have crafted to emulate the top level master state machine. For example, random selection of relocatable partial bitstreams and locations are implemented in C and transmitted to the SPREAD engine between encryption operations. Note that the emulation approach reduces the complexity of the task faced by an adversary in attacking the SPREAD implementation, and therefore, the
Chapter 4. Experiments

results presented in this thesis are optimistic from the standpoint of the adversary.

The SAKURA-X FPGA board is shown in Fig. 4.2(a). In our experiments, the SAKURA-X is first programmed with one of the versions of the SPREAD engine that we describe in the following section. A C program then applies 30,000 randomly generated plaintexts and coordinates with a LABVIEW program to collect a power trace for each plaintext. For the reference design associated with each version, the C program makes no changes to the reconfiguration regions. For the versions incorporating a countermeasure, the C program periodically instructs the FPGA, through the USB interface, to reprogram a randomly selected region with a randomly selected partial bitstream. The srand() function in C is used to generate the random numbers.

![SAKURA-X board](image1)

![TEK TDS7254 O-scope](image2)

Figure 4.2: SAKURA-X board used in the experiments and example power trace from round 1 of the AES encryption engine.

The interval between reprogramming operations is typically set to occur after every 32 plaintext encryptions. The justification for this interval is given as follows. The FPGA carries out encryptions at a clock frequency of 50 MHz. Therefore, the 10 rounds associated with a complete encryption operation is 200 ns (20 ns * 10 rounds). The amount of time to reprogram a region from ICAP is approx. 100 us. During
Chapter 4. Experiments

this interval, the FPGA can perform 500 encryptions (5 encryptions/us * 100 us). In our experiments, we repeatedly encrypt each plaintext 16 times to reduce the noise in the collected power trace. Therefore, 32 plaintexts * 16 encryptions/plaintext yields 512 total encryptions, which is slightly greater than the number that can be collected by the adversary while the architecture remains fixed under one particular configuration.

We use the same set of plaintexts in each experiment to enable an apples-to-apples comparison among the different versions. An example power trace is shown in Fig. 4.2(b), where the power transient associated with the first two rounds is captured and displayed on the oscilloscope. The oscilloscope is set to collect a 500 ns time interval at a resolution of 100 ps/point, yielding power trace waveforms with 5000 points.

4.1.1 Attack Model

We apply CPA to the 30,000 power traces obtained from each of the SPREAD versions. The attack point in the analyses is the output of the SBOX’s in Round 1 (see Fig. 4.2). The power model for CPA is constructed based on Hamming weight where we first compute the expected output of the SBOX for each key guess under each plaintext. Hamming weight simply counts the number of 1’s on the SBOX outputs. The power model is represented as a list of integers between 0 and 8, one for each of the plaintexts. Therefore, for each of the 16 SBOX’s, a set of 30,000 Hamming weights is computed for each key guess yielding a set of 256 lists.

CPA attacks each SBOX, one at a time, and uses Pearson’s correlation coefficient (PCC) to evaluate each of the 256 power models, which is defined by Eqn 4.1 [32]. Here, $X_i$ represents one of the power model values for a plaintext $i$ and $Y_i$ is a power trace voltage value corresponding to that plaintext. $X$ and $Y$ represent the means
Chapter 4. Experiments

of the power model values and power trace values, respectively. Note that Pearson’s correlation coefficient is sensitive to the mean values of the power traces. In order to eliminate dilution of the PCC that is introduced by temperature-related vertical shifts in the power traces, we pre-process the power traces by subtracting the mean of each power trace from the individual points in that power trace. This normalization process centers each power trace over 0 and removes the bias and corresponding dilution of the PCCs that would otherwise occur.

\[
PCC_i = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}}
\]

(4.1)

The power trace itself is composed of 5000 voltage values, so the PCC calculation is typically repeated at multiple different time points over the region of interest in the power traces. In our experiments, we repeat this analysis for each point in the range between 10 ns and 70 ns, inclusively. As indicated earlier, the oscilloscope sampling resolution is 100 ps/point, so the CPA analysis is applied 601 times. From Figure 9c, the 10 ns to 70 ns time interval includes the entire round 1 power transient. The CPA algorithm simply selects one of the 256 power models with the largest or smallest PCC, that is, the PCC closest to –1 or 1. Given that each power model is associated with a specific key guess, CPA predicts the key in this fashion one byte at a time, that is, this entire analysis is repeated 16 times, once for each key byte.

In order to portray the results of an entire analysis in a single graph, we develop a metric called PCC difference. The PCC difference metric captures the most important information from an evolutionary graph in a single value, and is defined using Eqn 4.2.

\[
PCCDifference = (PCC_{\text{correct\_key}} - PCC_{\text{largest\_incorrect\_key}})
\]

(4.2)
Chapter 4. Experiments

4.2 Experimental Results

4.2.1 SPREAD Versions

The nature of the approach taken by SPREAD presents a wide range of opportunities for implementing countermeasures. Fig. 4.3 shows a block level diagram of the AES engine, illustrating components of the data path which are the target of the countermeasure strategies. As discussed earlier, the reconfiguration regions shown in Fig. 3.6 are used to contain the logic associated with the diversified SBOX components, and constitute the heart of the proposed countermeasures investigated in this work. However, as discussed further below, the dat register shown in Fig. 4.3 is a contributor to the information leakage associated with the AES engine. Therefore, several of the versions of SPREAD attempt to address this leakage, either by adding jitter to the clock input to these registers or by including 8-bit chunks of the dat register in the SBOX reconfiguration regions themselves.

We break down the SPREAD versions according to the following categories. All versions include all the components of the proposed countermeasure described earlier, i.e, the 18 reconfiguration regions, the 3-to-1 MUX'ing scheme and the redundant copies of the SBOX.

- V4: A version that places only the SBOX combinational logic in reconfiguration regions.

- V5: A version the incorporates a 1-bit clock jitter scheme for each of the registers associated with the dat register in the static top level design. The 2-to-1 MUX can be configured on-the-fly to introduce 1 of 2 possible delays to the clock inputs of each of the dat registers independently. A schematic of the clock jitter circuit is shown in Fig. 4.4. The nonce bit is provided by the
Chapter 4. Experiments

Figure 4.3: Block diagram of the AES engine showing the datapath.

C program over the USB interface before the plaintext encryption operation begins.

- V6: A version identical to V5 but which incorporates a 2-bit clock jitter scheme, which enables 1 of 4 possible delays to the clock inputs of the dat registers.

- V11: A version in which 8-bit chunks of the dat register are synthesized along with the diversified SBOX netlists within each of the 18 reconfiguration regions. Therefore, the dat register shown in Fig. 4.3 is subsumed into the 16 SBOXs of the datapath.
Chapter 4. Experiments

- V12: A version with the clock jitter circuit inside the SBOX reconfiguration region with the 8-bit chunks of the dat register. (NOTE: We have not been able to get this version to synthesize properly, and is mentioned here only as a future experiment).

![Figure 4.4: Circuit used to introduce delay (jitter) to the clock inputs of the dat registers in the datapath component of the AES engine.](image)

4.2.2 CPA Evolutionary Graphs

Figs. 4.5 and 4.6 provide an illustration of the CPA attack technique as a *evolutionary* graph. The curves shown in the figures are obtained from the CPA method applied to the power traces from version V6, Byte 7. The curves in Figs. 4.5 are obtained from the reference design while those in 4.6 are obtained with the V6 countermeasure enabled. Each graph plots 256 superimposed curves, one for each key guess. The data points that define the curves are Pearson correlation coefficients (PCCs), given by Eqn 4.1, which are computed by correlating a voltage measurement (a point) in the 30,000 power traces with the power model data described earlier. The point chosen in the power traces is determined by repeating the PCC calculation over a range of points (in our experiments, the region between 10 and 70 ns from 4.2), and then selecting the point that produces the highest (or lowest) correlation. The black curves are associated with incorrect key guesses while the red curve is associated with the correct key guess.
Chapter 4. Experiments

The x-axis plots, on a log scale, the PCCs as larger numbers of power traces are added to the CPA calculation, beginning with 256 power traces ($\log_2(2^8)$), then 362 ($\log_2(2^{9.5})$), 512 $\log_2(2^9)$, etc. on the left and ending with $\log_2(2^{14.5})$, $\log_2(30000)$ on the right. This type of plot shows the evolution of the PCCs as uncorrelated noise averages toward zero and correlated signal behavior plateaus and emerges as the dominant component. It is clear that after approximately 16,000 traces that the curve associated with the correct guess (the red waveform) emerges as dominant both with and without the countermeasure.

![Evolutionary plot associated with Byte 7 of the reference design for version V6.](image)

**Figure 4.5:** Evolutionary plot associated with Byte 7 of the reference design for version V6.

None of the countermeasures implemented in the 4 versions reported on in this thesis succeeded in making the curves associated with the correct key guess rank at positions other than the top position for any of the 16 bytes. Therefore, we intend to continue to explore alternative architectures, such as those mentioned above for V12, as a means of providing protection against CPA attacks.
Despite the lack of success in finding an effective countermeasure, we are able to show that all four versions improved the SCA resistance against CPA attacks, and therefore the SPREAD technique represents a promising solution to this problem. The evidence that reflects this claim is presented in Fig. 4.7. Here, the bars in the bar graph each measure the sum of the differences between the PCC of the correct key guess and the PCC of the largest incorrect key guess. Given that the correct key guess was always ranked number one both the reference and countermeasure designs, the sums are all positive. However, the magnitude of the sums differ significantly between the reference design and the design incorporating the corresponding countermeasure. For example, the sum of the differences for the reference design of V6 are more than twice as large as the sum of the differences for the countermeasure. In fact, the countermeasures of all four versions provide improved SCA resistance over the reference design which is reflected by the smaller bars.
Chapter 4. Experiments

In an attempt to determine the limit of the effectiveness of the countermeasure, we ran a special experiment for V11 which reprogrammed all 16 SBOX locations in the SPREAD engine between every encryption. The blue bar in Fig. 4.7 portrays the sum of the differences associated with this experiment. The reprogramming strategy in this experiment represents the maximum level of signal obfuscation that is possible. The fact that it is still possible to deduce the correct key indicates that our proposed countermeasures are not addressing all sources of leakage present in the AES encryption engine, or the diversity of the SBOX implementations are not sufficient at eliminating all signal correlations within the SBOXs themselves. We are looking at alternative analysis methods, such as TVLA, to determine the primary source of the leakage in the implemented versions.

**Figure 4.7**: Comparison of average reduction in signal correlation for all 4 versions of the proposed SPREAD design relative to the levels of signal correlation present in the reference designs of each version.
Chapter 5

Conclusion

In conclusion, this thesis work introduced a novel architecture that aimed to make side channel attacks (SCA), namely correlation power analysis (CPA), impractical using a moving target approach. This research has not succeeded in preventing the secret key from being discovered using CPA. This leads us to believe that there are other sources of leakage that also need to be addressed within the AES engine, or that the diversity introduced in the SBOX implementations was not sufficient to eliminate all signal correlations within the SBOXs. The proposed architecture however has managed to reduce correlations leveraged by CPA as evidenced by the experimental results. This is a step in the right direction and opens avenues for future work to achieve the expected result. One technique that will be introduced in future versions of this work will be wave dynamic differential logic (WDDL) which, combined with the proposed architecture, we believe will provide enough CPA resistance to prevent the adversary from extracting the secret key.
References


References


References


References


[33] R. Zamacola, A. García Martínez, J. Mora, A. Otero and E. de La Torre, ”IMPRESS: Automated Tool for the Implementation of Highly Flexible Partial Reconfigurable Systems with Xilinx Vivado,” 2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Cancun, Mexico, 2018