2014

Intentional and Unintentional Side-Channels in Embedded Systems

Georg Tobias Becker
University of Massachusetts - Amherst

Follow this and additional works at: https://scholarworks.umass.edu/dissertations_2
Part of the Electrical and Computer Engineering Commons

Recommended Citation
https://scholarworks.umass.edu/dissertations_2/1

This Open Access Dissertation is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.
INTENTIONAL AND UNINTENTIONAL SIDE-CHANNELS IN EMBEDDED SYSTEMS

A Dissertation Presented
by
GEORG TOBIAS BECKER

Submitted to the Graduate School of the University of Massachusetts Amherst in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

February 2014

Electrical and Computer Engineering
INTENTIONAL AND UNINTENTIONAL SIDE-CHANNELS IN EMBEDDED SYSTEMS

A Dissertation Presented

by

GEORG TOBIAS BECKER

Approved as to style and content by:

____________________________
Christof Paar, Chair

____________________________
Wayne P. Burleson, Member

____________________________
Kevin Fu, Member

Prof. Christopher V. Hollot, Department Chair
Electrical and Computer Engineering
ACKNOWLEDGMENTS

It is finally done. After four years of being a PhD student (and many more being an undergraduate and master student) I realize while writing these lines that I am no student any more. I could even call myself “Dr. Becker” — strange. Of course, I have not done it all by myself. I have many people to thank for their help and support.

First of all, I want to thank my parents for their great support throughout my educational career and life in general. I can count myself lucky having such supportive and caring parents! Vielen Dank!

Next, it is time to thank my advisor Christof Paar for making this PhD possible. I especially want to thank him for his encouragement and his oversight of “hot topics”. I also want to thank my second advisor Wayne Burleson for his support and guidance and his help in becoming (at least partially) a hardware guy. I can now call myself without a guilty conscience an electrical and computer engineer. Many thanks also to my third committee member Kevin Fu for many interesting discussions in the SPQR meetings.

Being a member of a lab makes the PhD much more enjoyable. Therefore I have to thank all current and former members of the VCSG group. Special thanks deserves “meine Leidensgenossin” Gesine as well as Vikram and Ibis. I also want to thank “my second lab” in Bochum (both EMSEC and SHA). It has always been fun to come to Bochum and, believe it or not, I have even learned a lot during my visits. Many thanks also to the computer science members of the SPQR group.

This thesis would not have been possible if I would not have had help in my research. I therefore want to thank all of you who have helped in my research, espe-
cially my co-authors: Wayne Burleson, Tim Güneysu, Markus Kasper, Ashwin Lakshminarasimhan, Lang Lin, Oliver Mischke, Amir Moradi, Christof Paar, Francesco Regazzoni, Sudheendra Srivathsa, Daehyun Strobel and Vikram B. Suresh.

I would also like to thank CRI for a great internship as well as the students of my Security Engineering class for their patience with me (I really hope you enjoyed the class as I did). Last not least I want to thank my siblings, my “old friends” in Germany, as well as my “new friends” in the US for helping me keeping sane. Due to you guys, the last four years were actually quite fun :)}
ABSTRACT

INTENTIONAL AND UNINTENTIONAL SIDE-CHANNELS IN EMBEDDED SYSTEMS

FEBRUARY 2014

GEORG TOBIAS BECKER
B.Sc., RUHR-UNIVERSITY BOCHUM
M.Sc., RUHR-UNIVERSITY BOCHUM
Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Christof Paar

Side-channel attacks have become a very important and well-studied area in computer security. Traditionally, side-channels are unwanted byproducts of implementations that can be exploited by an attacker to reveal secret information. In this thesis, we take a different approach towards side-channels. Instead of exploiting already existing side-channels, they are inserted intentionally into designs. These intentional side-channels have the nice property of being hidden in the noise. Only their implementer can make use of them. This makes them a very interesting building block for different applications, especially since they can also be implemented very efficiently. In this thesis, techniques to build intentional side-channels for embedded software designs, RTL level hardware designs, as well as layout level hardware implementations are presented. The usefulness of these techniques is demonstrated by building efficient side-channel based software and hardware watermarks for intellectual property
protection. These side-channel based watermarks can also be extended to be used as a tool to detect counterfeit ICs, another problem the embedded system industry is facing. However, intentional side-channels also have malicious applications. In this thesis, an extremely stealthy approach to build hardware Trojans is introduced. By only modifying the IC below the transistor level, meaningful hardware Trojans can be built without adding a single transistor. Such hardware Trojans are especially hard to detect with currently proposed Trojan detection mechanisms and highlight not only the fact that new Trojan detection mechanisms are needed, but also how stealthy intentional side-channels can be.

Besides intentional side-channels, this thesis also examines unintentional side-channels in delay based Physically Unclonable Functions (PUFs). PUFs have emerged as an alternative to traditional cryptography and are believed to be especially well suited for counterfeit protection. They are also often believed to be more resistant to side-channel attacks than traditional cryptography. However, by combining side-channel analysis with machine learning, we demonstrate that delay based PUFs can be attacked, using both active as well as passive side-channels. The results not only raise strong doubt about the side-channel resistance and usefulness of delay based PUFs, but also show how powerful combining side-channel analysis techniques with machine learning can be in practice.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xii</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1. MOTIVATION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Organization and Contribution of the Thesis</td>
<td>4</td>
</tr>
<tr>
<td>2. BACKGROUND</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Side-Channel Analysis</td>
<td>8</td>
</tr>
<tr>
<td>2.1.1 Difference-of-Mean</td>
<td>9</td>
</tr>
<tr>
<td>2.1.2 Correlation Power Analysis</td>
<td>10</td>
</tr>
<tr>
<td>2.2 IP-Protection</td>
<td>12</td>
</tr>
<tr>
<td>2.2.1 Hardware Watermarks</td>
<td>14</td>
</tr>
<tr>
<td>2.2.2 Software Watermarks for Embedded Devices</td>
<td>17</td>
</tr>
<tr>
<td>2.3 Hardware Trojans</td>
<td>18</td>
</tr>
<tr>
<td>2.3.1 Hardware Trojan Design</td>
<td>20</td>
</tr>
<tr>
<td>2.3.2 Hardware Trojan Detection</td>
<td>21</td>
</tr>
<tr>
<td>2.4 Physical Unclonable Functions</td>
<td>23</td>
</tr>
<tr>
<td>2.4.1 Side-Channel Resistance of Physical Unclonable Functions</td>
<td>25</td>
</tr>
</tbody>
</table>
3. SIDE-CHANNEL BASED HARDWARE WATERMARK ........ 28

3.1 Watermark Design .................................................. 28
   3.1.1 Spread Spectrum Based Watermark ........................... 29
      3.1.1.1 Embedding the Watermark .............................. 30
      3.1.1.2 Detecting the Watermark .............................. 31
      3.1.1.3 Experimental Results ................................. 32
   3.1.2 Input-Modulated Watermark ................................. 33
      3.1.2.1 Embedding an Input-Modulated Watermark ............ 34
      3.1.2.2 Experimental Results ................................. 35
   3.1.3 Proof-Of-Ownership ......................................... 35

3.2 Watermark Robustness ............................................. 36
   3.2.1 Reverse-Engineering Attack ................................. 37
   3.2.2 Raising the Noise ........................................... 37
   3.2.3 Transmission of an Inverse Watermark Signal ............ 38

3.3 Counterfeit Protection ............................................ 40

4. SIDE-CHANNEL BASED SOFTWARE WATERMARK ........... 42

4.1 Software Watermark Design ....................................... 43
   4.1.1 Implementation ............................................... 44
   4.1.2 Watermark Verification ..................................... 46
   4.1.3 Triggering ..................................................... 48

4.2 Robustness and Security Analysis of the Software Watermark .... 50
   4.2.1 Reverse-Engineering Attack ................................ 51
   4.2.2 Code-Transformation Attacks ............................... 52
   4.2.3 Side-Channel Attacks ...................................... 56

4.3 Proof-of-Ownership ............................................... 58
4.4 Detecting Software Theft Without a Watermark ................ 59

5. SIDE-CHANNEL BASED HARDWARE TROJANS ............... 62

5.1 Dopant Trojans .................................................... 63
   5.1.1 Design of Dopant Trojans .................................. 64
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 Performance of the Trojan-free and Trojan iMDPL-AND gate for different input patterns and a load capacitance of 5.4fF. $A$ and $B$ depict the unmasked inputs to the iMDPL-AND gate and $M$ the mask bit. The column “0 → 1” shows the propagation delay of the evaluation phase and “1 → 0” the propagation delay of the precharge phase in picoseconds. “Risetime” represents the risetime of either $Y_m$ or $\bar{Y}_m$ during the evaluation phase.</td>
<td>80</td>
</tr>
<tr>
<td>6.1 Example correlation coefficients ($cc$) and corresponding noise levels, depicted as $\mathcal{N}(\mu_N, \sigma_N^2)$ and the number of corresponding noise register, of CPA attacks on different architectures: a PIC16F886 microcontroller that is used in an RFID access control system [62], the Yubikey One-Time Password Token that uses AES [62], the DS2432 and DS28E01 SHA-1 HMAC protected EEPROM from Maxim Integrated [62], Virtex-4 and Virtex-5 bitstream encryption based on AES-256 [57], an AES-128 implementation on a 22nm Kintex-7 FPGA [37], and an EM attack on the contactless Mifare Desfire smartcard (potentially with some side-channel countermeasures) [8]. Please note that the noise levels are only approximations based on the CPA values provided by the cited papers and are only meant to give the reader a rough idea of the expected noise levels in a side-channel attack.</td>
<td>103</td>
</tr>
<tr>
<td>6.2 Required number of challenges for different noise levels to achieve an accuracy large enough to find a string match with an 80-Bit HD power model. With such a string match a second machine learning algorithm achieved accuracies beyond 99%.</td>
<td>112</td>
</tr>
<tr>
<td>6.3 Required number of challenges for different noise levels to achieve an accuracy large enough to find a string match with a 1-Bit HW power model. With such a string match a second machine learning algorithm achieved accuracies beyond 99%.</td>
<td>112</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Architectural view of Intel’s CE4100 System-On-Chip that is used in Smart-TVs.</td>
<td>15</td>
</tr>
<tr>
<td>3.1</td>
<td>Diagram of the implemented spread-spectrum based watermark</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>Analysis of the spread spectrum based watermark while the AES core was idle and waiting for the next plaintext (a) using the leakage of 1000 clock cycles, (b) over the number of clock cycles</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>Analysis of the spread spectrum based watermark while the AES core was constantly encrypting (a) using the leakage of 250,000 clock cycles, (b) over the number of clock cycles</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Diagram of an input-modulated watermark that consists of an example of a combination function and a leakage circuit</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Detection of a spread spectrum based watermark that was counterbalanced by an inverse watermarking signal. We performed the same analysis as described in 3.1.1.2 while the AES core was idle (a) using the leakage of 10,000 clock cycles, (b) over the number of clock cycles</td>
<td>39</td>
</tr>
<tr>
<td>4.1</td>
<td>Example of a side-channel based software watermark that was inserted into the key expansion of an AES algorithm. The combination function as well as the leakage function is realized by adding a few additional assembly instructions into the code. The internal state needs to be changing in a predictable way. In this watermark the first two bytes of the plaintext were used as the internal state. The watermark constant is a fixed value that can be chosen by the watermark owner and forms together with the internal state the input to the combination function.</td>
<td>44</td>
</tr>
<tr>
<td>4.2</td>
<td>The result of the side-channel analysis plotted (a) against time and (b) with respect to different hypotheses, where hypothesis number 100 is the correct one</td>
<td>47</td>
</tr>
</tbody>
</table>
4.3 The results of the side-channel analysis with respect to number of measurements. Even with less than hundred measurements the correct hypothesis can be clearly detected. .......................... 48

4.4 Figure showing a CPA with 1,000 measurements to detect the watermark in two implementations with random-delay countermeasures. In both figures peak extraction and a pattern based alignment method were used. In (a) random delays are introduced by pseudo-randomly triggering a timer interrupt every 1-128 clock cycles. In (b) the side-channel countermeasure improved floating mean was added to the watermarked AES. In both cases the watermark is clearly detectable. .................. 55

4.5 Side-channel watermark that transmits an ID. Positive correlation peaks indicate that a ‘1’ is being transmitted, negative correlation peaks indicate a ‘0’. In this Figure we can see how the hexadecimal string “E926CFFD” is being transmitted. ................. 60

5.1 An unmodified inverter gate (a) and a Trojan inverter gate with a constant output of $V_{DD}$ (b). ............................... 65

5.2 Overview of Intel’s RNG design. An Entropy Source (TRNG) generates truly random numbers whose entropy is monitored by the Online Health Test (OHT). The random numbers are then fed to a digital random bit generator (DRBG) consisting of a Conditioner and a Rate Matcher. The Conditioner is used to periodically reseed the Rate Matcher which provides the output RnRand of the RNG. The correct functioning of the RNG is checked at each power up using the Build-In Self Test (BIST). The Trojan is inserted into the 256 bit state of the Rate Matcher. ........ 68

5.3 Layout of the Trojan DFFR_X1 gate. The gate is only modified in the highlighted area by changing the dopant mask. The resulting Trojan gate has an output of $Q = V_{DD}$ and $QN = GND$. ............. 71

5.4 Schematic of an iMDPL-AND gate consisting of two Majority gates, a detection logic and an SR-latch stage[65]. .......................... 76

5.5 Schematic of the Trojan-free and Trojan AOI222_X1 gate configured as a 3-input not-majority gate. .............................. 77
5.6 On the left (a) the layout of the unmodified AOI222_X1 gate and on the right (b) the Trojan AOI222_X1 gate is shown. In the Trojan gate the p-MOS transistors in the upper left active area have been shorted with the n-well by replacing the p-implant with n-implant. The strength of the remaining p-MOS transistors in the upper right active area have been reduced by decreasing the p-implant in this area. ................................................................. 78

5.7 1-Bit CPA on (a) the Trojan design and (b) the Trojan-free design using the Trojan power model with the evaluation phase starting at 0ns and the precharge phase starting at 15ns. The correct key is shown in black and the false keys are shown in gray. The correlation for the correct key in the Trojan design goes up to 0.9971. ................................................................. 81

5.8 8-Bit HW CPA attack on (a) the Trojan-free design and (b) the Trojan design with the precharge phase starting at 15ns. The correct key, highlighted in black, can be distinguished. However, the correlation coefficient of the attack for both the Trojan and Trojan-free is the same. ................................................................. 82

5.9 Figure (1) shows the result of a MIA attack with a 1024 bin histogram method and an 8-bit HW distinguisher for the Trojan design. The correct key, highlighted in black, never reaches the maximum for any time period and therefore the attack is unsuccessful. On the right the results of a 1-bit CPA on the first bit of the SBox output of the Trojan design is shown. Again the correct key, highlighted in black, can not be distinguished from false keys at any time instance. ................................................................. 83

5.10 Principle of the Trojan side-channel [50]. A Trojan side-channel $c$ that leaks out the secret key $K$ is inserted into the power consumption of the device. Only the attacker knowing the Trojan secret can recover the key $K$ from $c$. An evaluator on the other hand cannot distinguish $c$ from noise. ................................................................. 87

6.1 Schematic of an n-bit Arbiter PUF. ................................................................. 92
6.2 The controlled PUF design. An 80-bit master challenge is applied to
the controlled PUF from which 80 individual sub-challenges are
derived using the challenge generator. These 80 sub-challenges are
applied to the 128-bit Arbiter PUF and the 80 PUF responses are
stored in a shift register. This 80-bit string is hashed using a
cryptographically secure one-way function and the resulting 64-bit
hash value is provided as the final response of the controlled
PUF.

6.3 Two power traces of an 128-bit Arbiter PUF for two different
challenges, one challenge with a response of 1 and one with a
response of 0.

6.4 The correlation of responses with different accuracies with 2000
simulated power traces of a 128-bit Arbiter PUF.

6.5 Result of a CMA-ES with a 1-bit HW power model, 150k challenges
and a noise of $N(0, 25)$ which is equivalent to 100 switching
registers.

6.6 Result of a CMA-ES with a 1-bit HD power model, 150k challenges
and a noise of $N(0, 25)$ which is equivalent to 100 switching
registers.

6.7 Result of a CMA-ES with an 80-bit HD power model, 150k challenges
and a noise of $N(0, 25)$ which is equivalent to 100 switching
registers.

6.8 Relation between the correlation coefficient and the prediction
accuracy for the Hamming weight power model as well as the
Hamming distance power model. For this simulation 1 million
random response bits were used and no noise was added.

6.9 Result of of 100 runs of an 80-bit CCMA-ES attack with different
levels of noise. The noise level is expressed by the number of
randomly switching registers to achieve the same amount of
Gaussian noise. On the left the maximum achieved accuracy with
100 runs is shown while on the right the number of runs that
achieved an accuracy high enough to find at least one string
match is shown.

6.10 The number of needed strings so that the probability of a match is at
least 50%.
6.11 The delay difference in pico seconds between the top and bottom signal after the last stage for 49k traces. Colored in blue are the delay differences of all traces and in black are the delay differences for the traces whose output flipped when the supply voltage was changed from 1.1V to 1V and 1.2V. .......................... 114

6.12 The result of 100 runs of the CMA-ES Fault attack with 8k traces and +/- 0.1V supply voltage variation without additional noise. On the left the accuracy of the resulting PUF models are shown. On the right side the number of 80-bit strings that were correctly predicted by the PUF models are shown. .......................... 118

6.13 The result of 100 runs of the CMA-ES Fault attack with 45k traces and +/- 0.1V supply voltage variation with 20% faulty responses. .......................... 118

6.14 Result of a Fault CMA-ES attack with 200 independent runs for different levels of overall-noise and for different numbers of challenges. On the left the highest achieved model accuracy is shown while the right figure shows the number of runs that had at least one string match. .......................... 119
CHAPTER 1
MOTIVATION

1.1 Introduction

Embedded systems are being increasingly used in a wide range of applications. Embedded systems range from smart-phones with GHz multi-core processors, over intelligent refrigerators and coffee makers to medical devices or smart cards for payment systems. The general trend towards more and more devices having computation power was first summarized as the trend of *ubiquitous and pervasive computing*. Nowadays the term *Internet of Things* is increasingly used instead. This new term indicates that most devices will not only have some kind of computation and communication ability, but that they will all be connected through a single huge network — the Internet. Many other notations come along with this trend, such as the *smart* notations like *smart cities*, *smart grids* or *smart house*. These ideas have in common that the deployment of a network of embedded devices with some kind of sensor, communication and computation ability increases the efficiency of larger systems — making them *smarter*.

The trend towards a large number of interconnected embedded systems creates many opportunities. However, there also come many challenges with this trend. As more applications enter the embedded system market, more products have to be developed in a very competitive industry. The time-to-market needs to be short to be able to keep up with the innovations, while at the same time the costs need to be kept low to remain competitive. Hence, embedded systems need to be designed faster and more efficiently while the complexity of the designs grows. This results in the need of
an increased re-use of designs. Companies cannot afford to develop everything on their own and from scratch, but need to obtain some functionality from third parties, e.g., in form of software for embedded systems or IP-cores for hardware implementations. This interchanging of intellectual property between companies raises the question of how it can be efficiently protected. Intellectual property protection is important especially in an industry that heavily relies on constant innovation.

The increasing complexity of embedded systems as well as the increased use of IP-cores from third-parties also opens new doors for hardware Trojans. Hardware Trojans are malicious alterations of the physical design that compromise the security or safety of the attacked device. Hardware Trojans are an increasing concern in government and industry and has gained increasing attention in the research community [2, 3, 6, 49]. A major concern is that a foreign government might put pressure on their IT industry to provide hidden backdoors to specific products manufactured in their country. However, so far there is only anecdotal evidence that hardware Trojans have already been deployed by governments. While a lot of research has focused on detecting hardware Trojan, there still seems a gap in the knowledge of how hardware Trojans can be implemented efficiently. A solid understanding of how hardware Trojans can be build is a requirement to judge the different Trojan detection mechanisms. In this thesis, new techniques to build extremely stealthy hardware Trojans are presented that will help to close this gap and enable researchers to build more efficient detection mechanisms that can also thwart these new types of hardware Trojans.

The trend towards embedded systems also introduces new security related problems. Unlike personal computers (PCs) and servers, embedded systems are no longer solely located in areas with restricted physical access such as homes or workplaces. Instead, they are often carried around or might be located at publicly accessible spots. Hence, when evaluating the security of these devices, it needs to be assumed that an attacker might have physical access to the device he is trying to attack. And this
leads to many new attack vectors such as side-channel attacks, probing attacks or fault-injection attacks.

Besides intellectual property protection and hardware Trojans this dissertation focuses on an aspect of hardware security that is one of the major concerns and research areas in hardware security: side-channel analysis. In a side-channel analysis an attacker exploits the fact that an embedded device is not a black-box that obtains defined inputs and only produces defined outputs. Instead, every physical system that performs some kind of computation will inevitably leak additional information over physical channels, such as the power consumption, the required execution time, or the thermal profile. In side-channel analysis, these physical properties are first measured and then exploited to derive additional information about the embedded system. Such information could, for example, be the secret key that is used in an encryption algorithm.

Unlike most previous work, this dissertation does not only look at side channels as an unwanted byproduct of a physical implementation. Instead, side channels are introduced intentionally into embedded systems. Interestingly, these intentional side channels can be used in constructive ways, e.g., for intellectual property protection in embedded systems. Hence, this dissertation shows how a method that has been primarily used in a malicious way to attack systems can be used for protection instead. However, intentional side-channels can also be used maliciously by introducing techniques to build extremely small and stealthy hardware Trojans. The fact that intentional side-channel can be used for both, constructive and malicious applications, highlights the flexibility and usability of side-channels.

In addition to these intentional side-channels a novel way to combine side-channel analysis and machine learning algorithms is presented to attack delay based Physical Unclonable Functions (PUFs). PUFs have emerged as an alternative to traditional security. Due to process variations, every chip has slightly different characteristics.
Strong PUFs harvest these process variations so that every chip can be uniquely identified using a challenge-and-response protocol. However, the most promising string PUF, the Arbiter-PUF, can easily be attacked using machine learning algorithms if the attacker has access to challenge-and-response pairs. Nevertheless, Arbiter-PUFs have some very promising properties such as the fact that no secret key needs to be programmed and stored and it is often argued that PUFs have a high resistance against implementation attacks. However, as we will see in this dissertation, Arbiter-PUFs can be attacked by combining side-channel analysis with machine learning algorithms. Therefore this dissertation raises strong doubts about the assumption that PUFs are more resistant against implementation attacks.

1.2 Organization and Contribution of the Thesis

The three main contributions in this thesis:

1. Introduced an easy and reliable way to detect IP theft in embedded systems by using side-channel based watermarks for hardware as well as software implementations. Unlike most other watermarks, to detect these watermarks a verifier only needs to have physical access to the IC under test.

2. Developed extremely stealthy dopant-based hardware Trojans that are not detected by the hardware Trojan detection mechanisms proposed in the literature. Two case-studies were used to highlight their versatility and show that meaningful hardware Trojans can be build with this new Trojan technique. These new Trojans give valuable insight for developing new Trojan detection mechanisms.

3. By combining side-channel analysis with machine learning attack we demonstrated that delay based PUFs are not considerably more resistant against implementation attacks as it often believed. These new discoveries raise the
question how promising delay based PUFs really are to build secure challenge-and-response protocols in embedded systems.

All contribution have two things in common. From a technical perspective, they all use side-channel analysis in an unconventional way, i.e., by inserting the side-channels intentionally into the design\(^1\) or by combining side-channel analysis with machine learning attacks. From an application perspective, they are all directly related to problems arising from the embedded system development process, i.e., IP-Theft, hardware Trojans, and anti-counterfeiting.

The main application for constructive use of intentional side-channels in this work are watermarks for intellectual property protection. In Chapter 3 a robust hardware watermark technique is introduced that is inserted by adding only a few additional gates to a hardware design. The main advantage of a side-channel based hardware watermark compared to other hardware watermarking techniques is that side-channel watermarks can be inserted at the Hardware Description Language (HDL) level while they can be efficiently detected in a fabricated chip. Most watermarks previously proposed could only be detected at the same level they were inserted. Hence, many watermarks can not be detected efficiently after manufacturing, making them impractical for most scenarios. Therefore the introduced side-channel watermarks have a great potential in protecting the intellectual property of hardware designs.

The idea of side-channel based watermarks is extended towards embedded software designs in Chapter 4. These side-channel based software watermarks are the first software watermarks that are specifically designed for embedded applications. Unlike most proposed techniques, side-channel software watermarks can be embedded at the assembly level and do not need to be added in a higher programming language such as Java. Furthermore, no access to the program code or memory structure is needed.

\(^1\)The dopant-based hardware Trojan can be used to build intentional side-channels at the layout level. However, these dopant Trojans also have applications beyond side-channels.
to detect the watermark. The watermark can be detected by simply having physical access to a device that executes the suspected code. This provides a major advantage since in embedded systems the access to the program memory is often restricted, making them a very useful tool to detect software theft in embedded systems.

Chapter 5 examines how to efficiently build hardware Trojans. A novel hardware Trojan technique based on dopant modification is introduced, that can be used to build extremely stealthy Trojans. The major advantage of this new Trojan is the ease of insertion during chip manufacturing. Further, compared to insertion of additional transistors or modification of metal layer, modifying only the dopant layer makes the trojan significantly harder to detect using existing reverse engineering techniques. The technique of building hardware Trojans by manipulating the dopant is introduced in Section 5.1 followed by two case studies demonstrating the usefulness of this new type of Trojans. In the first case study, a hidden backdoor is inserted in the digital post-processing of a secure random number generator design derived from random number generator used in Intel’s Ivy Bridge processors. Secret keys generated with such a Trojan infested random number generator have a significantly lower entropy and hence are insecure. However, the Trojan infested design will still appear legitimate due to the fact that the design passes functional testing as well as statistical tests. In the second case study in Section 5.2 the idea of dopant-Trojans is used to create a hidden side-channel in an otherwise side-channel secure AES design to leaks out the secret key. This intentional side-channel is designed in a way that it does not reduce the side-channel resistance against the most common side-channel attacks, thereby making the resulting design still appear legitimate and side-channel resistant to an evaluator.

A brief summary of how to build side-channel based Hardware Trojans at the gate level is given in Section 5.3.
Physical Unclonable Functions have emerged as a new cryptographic primitive that can be used among other things for anti-counterfeiting and secure device authentications. In Chapter 6 passive as well as active side-channel attacks on the most promising strong PUF design, the Arbiter-PUF are described. These new attacks raise doubt on the assumption that PUFs are more resistant against implementation attacks than traditional cryptographic algorithms. After the idea of Arbiter-PUFs is explained in Section 6.1 the concept of a controlled PUF that is resistant to machine learning algorithms is introduced. This controlled PUF design serves as the target for the combined side-channel and machine learning attacks. In Section 6.2 a power side-channel attack attack on controlled Arbiter-PUF is demonstrated using a combination of correlation power analysis and evolution strategies. One argument why Arbiter-PUFs have a high resistance to implementation attacks that is often repeated is that any tampering with the IC will change the output behavior of the PUF. However, in Section 6.3 we will learn that the information of which responses change when the IC is being tampered with can actually be used by an attacker to accurately model the PUF. Hence, the property that tampering changes the output behavior of PUFs does not necessarily make PUFs more secure against implementation attacks but makes PUFs vulnerable to a combined machine learning and fault attack.

The findings in this thesis are summarized in Chapter 7 and possible future work is discussed.
2.1 Side-Channel Analysis

Traditionally, in a cryptographic security evaluation, encryption algorithms are treated as a black box: They only produce well defined outputs for well defined inputs while intermediate values never leave this black box. From a cryptographic perspective the encryption algorithm is secure if it is computationally infeasible for an attacker to compute the key even if he has unlimited access to any chosen plaintext-ciphertext pairs. Many cryptographic algorithms exist that are considered secure from a cryptographic perspective. However, in 1998 Kocher et.al. [47] showed that an actual implementation of an encryption algorithm does not behave like a black box. It turns out that every integrated circuit (IC) that performs the encryption leaks out additional information over so called side-channels. With the help of these side-channels, it is possible to compute the secret key of otherwise secure cryptographic algorithms.

The most commonly used side-channels for this type of attacks are the execution time, the power consumption or Electro Magnetics (EM). But other side-channels such as heat or even optical emission can be used as well. The power consumption of an IC is not constant but instead depends on the processed data. For example, a typical CMOS inverter has a large power consumption while it switches. But when the transistor does not switch, the power consumption is very small. Kocher et.al. exploited this fact to successfully recover the secret key of a DES encryption engine by measuring the power consumption and performing some statistical analysis on these
power traces. In the next Section, this type of attack which is called Differential Power Analysis (DPA), will be introduced. Several variants of the original attack have been proposed over the year. One important extension is the Correlation Power Analysis (CPA) which is mainly used in this thesis.

2.1.1 Difference-of-Mean

A Differential Power Analysis (DPA) is based on two key observations: (1) the power consumption of an encryption algorithm depends on the processed intermediate values and (2) these intermediate values depend on parts of the key and plaintext.

For example, assume that the power consumption of the target device is larger when an intermediate value \( v \) is \( v = 1 \) compared to \( v = 0 \). In a difference-of-mean attack, power measurements of the target device with many different inputs are taken and each power trace is sorted into two piles: One pile for traces with \( v = 1 \) and one pile for traces with \( v = 0 \). Since we know that the power consumption should be larger for \( v = 1 \), we expect the mean (average) power consumption of the first pile to be larger than the mean power consumption of the second pile. However, since the power consumption of the device depends on more factors than just this one intermediate value, it could be that some power traces in the first pile have actually a smaller power consumption than some traces in the second pile. But by averaging over many traces, the mean power consumption of the pile with the 1s should be larger than the mean power consumption of the pile with the 0s. Hence, the difference of mean between these two piles should be positive.

In an actually DPA attack the secret key is not known and hence we cannot compute the intermediate value \( v \). This problem is solved by guessing parts of the key and using these guessed keys to compute the intermediate values. If the correct key was guessed, we expect the same effect as discussed above — that the difference-of-mean is positive. However, if we guessed the wrong key then we are essentially
randomly sorting the power traces into two piles. If you sort a large amount of traces randomly into two piles, the mean power consumption of the two piles should be similar and hence the difference-of-mean should be small.

Guessing a complete cryptographic key correctly is computationally infeasible due to the extremely large key space. However, for the DPA attack to be successful an attacker does not have to guess the entire key at once. The targeted intermediate valued only depend on parts of the key, e.g. 8 bits of a 128-bit key. Therefore, the attacker only has to guess these 8 bits correctly. Since there are only 256 different 8-bit subkeys, an attacker can easily compute the difference of mean for all of these subkeys. The correct subkey has the largest difference-of-mean and can therefore be identified. After the first subkey has been determined, the attack is repeated for other parts of the key. Using this divide-and-conquer approach, one subkey after the other is revealed until the entire key is known.

2.1.2 Correlation Power Analysis

The correlation power analysis (CPA) is an extension of the DPA. The disadvantage of the difference-of-mean method is that the used power model is very limited. For example, a common power model for an intermediate value in a microcontroller is the a Hamming wight (HW) power model, i.e. the power consumption is proportional to the number of ones in the intermediate value. However, the difference-of-mean is a 1-bit distinguisher and therefore cannot efficiently make use of this multiple bit information.

A CPA on the other hand can be used with more complex power models $f$ such as the Hamming weight or Hamming distance model. In a CPA, for every hypothetical intermediate value $v_i$ a hypothetical power value $h_i$ according to the power model $f$ is computed, $h_i = f(v_i)$. If the correct power model and intermediate values are used, then the hypothetical power values $\tilde{h}$ should be similar to the measured power values.
\( \vec{x}, \) i.e., the power values \( \vec{h} \) have a linear relation with the measured values \( \vec{x} \). To test the linear relation between two signals, in this case the measured power values \( \vec{x} \) and the computed hypothetical power values \( \vec{h} \), the Pearson correlation coefficient can be used:

\[
p = \frac{\text{cov}(\vec{h}, \vec{x})}{\sqrt{\text{var}(\vec{h})\text{var}(\vec{x})}}
\]

with \( \text{var} \) indicating the sample variance and \( \text{cov} \) the sample covariance. One advantage of the correlation coefficient is that it is bound between +1 and −1, i.e., \(-1 \leq p \leq 1\). Furthermore, the correlation coefficient is directly related to signal-to-noise ratio. This makes comparing different attacks and measurement setups easier and also allows to compute the bound of expected correlation coefficient for false key guesses. For wrong key guesses the correlation coefficient follows a Gaussian distribution with a standard deviation of \( \sigma = 1/\sqrt{\#\text{traces}} \) and a mean of \( \mu = 0 \). This way a boundary for the maximum correlation coefficient \( p_{\text{wrong}} \) that can be expected for false key guesses with a probability of 99.99\% can be computed using: \[51\]

\[
|p_{\text{wrong}}| = \leq \frac{4}{\sqrt{\#\text{traces}}}
\]

Looking at the correlation coefficients of successfully performed attacks also helps to judge how much noise one can expect in practice. The sum of the various noise sources in a power analysis can be approximated with a Gaussian distribution \( \mathcal{N}(\mu_N, \sigma_N^2) \)[74]. Hence, the power consumption follows a random variable \( X \) with \( X = H + \mathcal{N}(\mu_N, \sigma_N^2) \), where \( H \) is the random variable of the assumed power model. In this specific case the correlation coefficient can be written as:

\[
p = \frac{\text{cov}(H, H + \mathcal{N}(\mu_N, \sigma_N^2))}{\sqrt{\text{var}(H)\text{var}(H + \mathcal{N}(\mu_N, \sigma_N^2))}} = \frac{\text{cov}(H, H) + \text{cov}(H, \mathcal{N}(\mu_N, \sigma_N^2))}{\sqrt{\text{var}(H)^2 + \text{var}(H)\text{var}(\mathcal{N}(\mu_N, \sigma_N^2))}}
\]

\[
= \frac{\text{var}(H)}{\sqrt{\text{var}(H)^2 + \text{var}(H)\sigma_N^2}}
\]
Hence, it is possible to compute the expected correlation coefficient for a given noise distribution $\mathcal{N}(\mu_N, \sigma_N^2)$. Similarly, it is also possible to derive the noise level from measured correlation coefficients. We will make use of this in Chapter 6 to provide estimates of how much noise can roughly be expect in a side-channel attack on PUFs. To do this, we use the results of successful CPA attacks on different systems to approximate the noise in these systems and use these values as a reference.

2.2 IP-Protection

Intellectual Property (IP) protection is a major concern in many different areas. Especially in a field driven as strongly by innovation as the IT industry, protecting the intellectual property is essential for a successful business. There are two separate IP related problems a company might face: On the one hand, their products could be subject to counterfeiting. In counterfeiting, a genuine product is reproduced by an unauthorized entity and sold under the label of the victim company. This could be a direct copy of the genuine product, a completely different product that is simply relabeled to resemble a product from this company, or a second-grade product that is relabeled as first-grade. Counterfeit chips have already become a serious problem for the IT industry and the reported incidents have grown significant over the last years [31]. Counterfeit chips not only cost the company whose chips are being counterfeited revenue, but can also cause serious damage for the companies that unknowingly embedded these chips into their products. These counterfeit chips are most often less reliable, have worse performance or are broken from the start. Hence, products with such counterfeit parts will likely be much less reliable and have a worse performance. This way counterfeit chips, that might just cost a few cents, can result in enormous costs due to possible product re-calls and customer satisfaction losses.

On the other hand, their design could be stolen and illegitimately used in a product by a different company. This IP-theft can be in the form of software or hardware IP-
cores that are illegally resold or reused by an unauthorized company or in the form of copying the idea of a design or patent infringements. Note that IP-theft could also happen by accident by companies that lose track of their license agreement. For example, an IP-core that was licensed only for a particular application might be reused in a different project by an engineer that is not aware of the specific license agreement. But in both cases, whether the IP is illegally used by accident or by malpractice, the owner of the company loses revenue. In this dissertation I will introduce efficient methods to detect the illegal copying and using of software or hardware IP-cores by means of building an intentional side-channels into the design that serves as a watermark. While the focus of this work is on detecting IP-theft, how intentional side-channels can be used to protect against some types of product piracy and counterfeiting is also addressed in Section 3.3. Another anti-counterfeiting solution that has been proposed in the literature is the use of strong PUFs. However, as we will see in Chapter 6, how to build secure strong PUFs is still an open research problem.

One method to protect against IP-theft is digital watermarking. Using digital watermarks is a wide-spread concept used in many other contexts, e.g., to protect the intellectual property of pictures, audio, or video data. The idea is to embed information, the digital watermark, into the data you want to protect in a way that it is difficult to remove the watermark without destroying the data. This way each copy of the data will also include the embedded watermark information. Watermarks are most often used in copyright protection systems for digital media to deter unauthorized copying. In this section the concept of digital watermarking for hardware designs as well as software designs is introduced.
2.2.1 Hardware Watermarks

The hardware design process is an expensive and time consuming task. The increasing complexity of embedded systems often prevents designing an entire chip from scratch if one wants to stay economical. Instead, parts of the design are reused from earlier developments or bought from other companies as so-called IP-cores (intellectual property cores). These IP-cores implement specialized functionality to be used as building blocks within integrated circuit designs. For example, Figure 2.1 shows the architectural view of an System-On-Chip design as it is used for Smart-TVs. As one can see, many different IP cores are combined in a single chip. IP-cores can be separated into soft and hard IP-cores. A soft IP-core consists of an implementation in a synthesizable register transfer language (RTL) such as Verilog or VHDL. It can be sold either directly in an RTL format or as a generic gate-level netlist. RTL IP-cores allow the user to adapt the licensed design to his application specific requirements, while this is cumbersome with a netlist IP-core. Nevertheless, netlist cores, just as RTL cores, are not restricted to a specific technology and can be ported to any process or foundry. Hard IP-cores on the other hand are physical designs that come as completely laid out function blocks that cannot be modified and are thus restricted to a specific technology. Analog circuits are usually sold as hard IP-cores.

The IP-core market has risen to a multi-billion dollar business, making it a valuable target for fraud and piracy. The threats of cloned products, IP theft, and copyright infringement necessitate semiconductor designers and manufacturers to implement countermeasures into their products. Several possible protection methods against illegal IP usage have been proposed in the past. One solution to prevent IP-core theft is to deliver encrypted IP-cores only. These IP-cores can then only be decrypted by the tools which synthesize the design and their plain source code will therefore never be seen by the customer [32]. However, this approach is logistically very difficult and does not prevent customers who legally bought the IP-core from illegally sharing or
Figure 2.1. Architectural view of Intel’s CE4100 System-On-Chip that is used in Smart-TVs. Due to its complexity, it is not economical to develop everything from scratch. Instead, many different IP cores are used to build the system.

reusing it for multiple designs. A promising solution proposed to counter the IP theft threat is the concept of watermarking IP-cores [41, 60, 61, 76].

The goal of hardware watermarks is to be able to check if a given IP-core has been used in a design. It therefore does not prevent IP-theft itself, but it enables the detection of this theft and can help the owner to prove the theft in court.

When adapting the digital watermarking concept to protect hardware designs, the threat to counter is unauthorized usage of IP-cores. The goal of the designer of a circuit is therefore to be able to distinguish if his design is used in a given integrated circuit (IC) or complete product.

The IC that needs to be tested is in most cases only available as a completely assembled and packaged chip. Therefore, watermarks that can be detected even after manufacturing are most useful. However, in many hardware watermarking schemes, the watermark is implemented to protect only the high level representation of a chip design, e.g., the digital VHDL or Verilog representation. This kind of watermark cannot be detected in synthesized products anymore.
When implementing a watermark-protected design, detecting unauthorized use is the most interesting motivation. However, it is also desirable to be able to prove towards a third party, e.g., a judge, that your design has been used. This goal is called proof-of-ownership and is necessary to perform legal actions against the discovered theft. We summarize the goals of IP watermarking schemes:

1. Detectability: Given an IC, the owner of an IP-core can examine whether or not his IP-core is used in the IC.

2. Proof-of-ownership: Given an IC, the owner of an IP-core can prove to a third party that his IP-core is used in the IC.

Consequently, attacking a watermarking scheme means to either violate the detectability or proof-of-ownership goal. Violating the detectability goal is to remove the embedded information of the watermark from the IP-core or to render it useless. Furthermore, in a successful attack the removal of the watermark must not destroy the functionality of the IP-core. Violating the detectability goal obviously also implies breaking the proof-of-ownership property — if a verifier cannot detect the theft he surely cannot proof the theft towards a third party. But violating the proof-of-ownership goal on the other hand does not automatically imply that the detectability goal is violated as well.

A short overview of different watermarking techniques for IP protection can be found in [5]. Concepts for watermarks have been proposed for many different levels of the hardware design process. One of the most popular schemes suited for IP-cores are the constraint-based watermarks introduced in [41, 60]. In these schemes hardware designs are tagged by defining additional design constraints which do not affect the functionality of the IP-core. One major drawback of constraint based watermarks is that the watermark can only be discovered at the same level of abstraction they were inserted, i.e., these watermarks cannot be detected in produced chips [5]. As in most
cases a verifier will not have access to the high abstraction levels of a suspicious integrated circuit, this type of watermarking is impracticable for many scenarios. Some hardware watermarks use side-channels for evaluation. The advantage of embedding a watermark in the characteristics of the power consumption of a circuit is that it can easily be detected even post-manufacturing, although being embedded at a high level of abstraction, e.g., as a code in a hardware description language like VHDL or Verilog. In [83] the watermark modulates the power consumption side-channel to transmit a signature by means of On-Off Keying (OOK) and Binary Phase Shift Keying (BPSK) during the reset phase of an FPGA. However, this approach is not stealthy, meaning that the watermark is visible not only to the evaluator but to everyone. This makes it comparably easy to remove the watermark. In [45] a watermark has been proposed that uses temperature modulation to transmit the watermark information.

2.2.2 Software Watermarks for Embedded Devices

Software plagiarism and piracy is a serious problem, which is estimated to cost the software industry billions of dollars per year [19]. Software piracy for desktop computers has gained most of the attention in the past. However, software plagiarism and software piracy is also a problem for companies working with embedded systems. For example, many customers require access to the embedded system libraries to test these libraries before purchasing them. However, once a company has provided these libraries, there the company has lost direct control over it. If the customer decides to not buy the library, the library cannot be taken back — the company simply has to trust the customer that he does not illegally use or distribute the library.

In the following we will focus on the unique challenge to detect software plagiarism and piracy in embedded systems. If a designer suspects that her code has been used in an embedded system, it is quite complicated for her to determine whether or not her suspicion is true. Usually she needs to have access to the program code of
the suspected device to be able to compare the code with the original code. However, program memory protection mechanisms that prevent unauthorized read access to the program memory are frequently used in today’s microcontrollers. Hence, to gain access to the program code of an embedded system these protection mechanisms would have to be defeated first. This makes testing embedded devices towards software plagiarism very difficult, especially if it needs to be done in an automated way. Furthermore, the person who is illegally using unlicensed code might apply code-transformation techniques to hide the fact that he is using someone else’s code. In this case detecting the software theft is hard even if the program code is known.

Software watermarks [82] enable a verifier to test and proof the ownership of a piece of software code. Most watermarks usually require access to the program code [34] or the data memory [81, 21] during the execution of the program to detect the watermark. But as mentioned before, access to the memory and the program code is usually restricted in an embedded environment. Most software watermarks proposed in the literature are used for higher programming languages such as Java and C++. But in embedded applications many programs are written in C or assembly. Furthermore, many previously proposed software watermarks have shown to be not very robust to code-transformations [34]. Therefore software watermarks that are suitable for PC applications might not be very suitable for embedded systems. The side-channel based software watermarks presented in this thesis on the other hand does not need access to the program code or memory since they can be detected using side-channel measurements. Side-channel based watermarks can also be inserted into assembly code, making them especially well suited for embedded applications.

2.3 Hardware Trojans

The increased use of out-sourcing and globalization in circuit manufacturing has given rise to several trust and security issues, as each of the parties involved potentially
constitutes a security risk. In 2005 the Defense Science Board of the US Department of Defense (DoD) published a report in which it publicly voiced its concern about US military reliance on ICs manufactured abroad [2]. One threat in this context is that malicious modifications, also referred to as hardware Trojans, could be introduced during manufacturing. There is the risk that chips with hardware Trojans could be introduced into the supply chain. The discovery of counterfeit chips in industrial and military products over the last years has made this threat much more conceivable. For instance, in 2010 the chip broker VisionTech was charged with selling fake chips, many of which were destined for safety and security critical systems such as high-speed train breaks, hostile radar tracking in F-16 fighter jets, and ballistic missile control systems [31]. Hardware Trojans can also be inserted during the design phase, e.g. by third party IP-cores, software tools, malicious employee, or by replacing legitimate IP cores by hacking into the companies server. All this raises the question of trust in the final chip, especially if chips for military or safety-critical civilian applications are involved.

The threat of hardware Trojans is expected to only increase with time, especially with the recent concerns about cyberwar, cf., e.g., [52, 72]. Recent revelations from former NSA contractor Edward Snowden about the extent of the activities of the NSA have gained widespread attention. There is still no hard evidence that hardware Trojan have been inserted into computer chips. However, the NY times states based on the documents provided by Snowden about NSA’s Sigint project that: [64]

“By this year, the Sigint Enabling Project had found ways inside some of the encryption chips that scramble information for businesses and governments, either by working with chipmakers to insert back doors or by exploiting security flaws, according to the documents.”

This shows that the threat of hardware Trojans is quite real. The trust in hardware has significantly suffered as can be seen by the move of OpenBSD to stop relying on hardware RNG generators, not because of a poor RNG design, but because the
OpenBSD community has lost its trust in the chip manufacturers [30]. Hence, it is important to have a more thorough understanding of hardware Trojans and hardware backdoors so that we can start to develop new methods to regain this lost trust.

Research efforts targeting hardware Trojans can be divided into two parts, one related to the design and the implementation of hardware Trojans, and one addressing the problem of detecting hardware Trojans. In this section contributions from both areas are summarized.

### 2.3.1 Hardware Trojan Design

There have been relatively few research reports addressing the question of creating (as opposed to defeating) hardware Trojans, with the first hardware Trojans published around 2008. Most proposed hardware Trojans consist of small to mid-size circuits which are added at the HDL level. For example, in [46] a hardware Trojan for a CPU was proposed that can grant an attacker complete control of the system from outside. The attacker can make arbitrary changes to the program code and can get unlimited access to the memory by simply sending a specific malicious UTP package to the processor. This Trojan shows how vulnerable systems become ones the root of trust — the hardware — is compromised. However, inserting such a Trojan which consists of a few hundred gates would be very challenging at layout-level and the additional gates can easily be detected using optical reverse-engineering. Another class of HDL-level Trojans are those which create a hidden side-channel to leak out secret keys by adding only a few additional gates [50]. Perhaps most of the Trojans proposed so far were shown at the annual hardware Trojan challenge by NYU-Poly, where students insert hardware Trojans into a target FPGA design with the goal of overcoming hardware detection mechanisms [67].

All these Trojans have in common that they are inserted at the HDL level. The attack scenario is here that a malicious circuitry is introduced into the design flow of
the IC. However, these Trojans are difficult to realize by a malicious foundry which usually only has access to the layout mask. In this context, finding the needed space and adding extra connections to place & route the Trojan gates can be unpractical. How realistic these Trojans are in a foundry-based attack model is therefore still unanswered.

A more realistic scenario for a foundry-based Trojan insertion are malicious modifications carried out at layout level. An example of such a Trojan are the Trojans proposed by Shiyanovskii et al. [73]. In this work the dopant concentration is changed in order to increase the effects of aging on the circuit, with the ultimate goal of reducing the expected lifetime of the device. However, these Trojans have limited usability, since it is hard to predict the exact time the ICs will fail and they can usually only serve as a denial-of-service type of Trojan.

2.3.2 Hardware Trojan Detection

Hardware Trojan detection mechanisms can be divided into post-manufacturing and pre-manufacturing detection mechanisms. The input to pre-manufacturing Trojan detection is usually the gate netlist or HDL description of the design under test. Pre-manufacturing Trojan detection tries to detect Trojans that have been inserted at the HDL level into the design flow, e.g., by third party IPs, design tools or untrusted employees. Usually the Trojan detection is based on functional testing or formal verification. There have been also proposals of how to defend rather than detect hardware Trojans at the HDL level. One approach is to replace part of the hardware design that was not covered by functional testing by software [36]. The other approach is to add redundancy and a control circuitry between untrusted IPs that will make Trojan activation based on counters and inputs difficult [78]. However, these proposed Trojan detection and prevention mechanisms cannot be applied to Trojans at the sub-gate level, such as the ones proposed in this thesis.
Post-manufacturing Trojan detection mechanisms primarily attempt to detect Trojans inserted during manufacturing. They can be divided into two categories based on whether or not they need a “golden chip” (also referred to as golden model). A golden chip is an IC which is known to not include malicious modifications. The standard approach proposed to detect layout-level hardware Trojans and to find a golden chip is the use of optical reverse-engineering. The idea is to decap the suspected IC and make photos of each layer of the suspected chip with e.g., a scanning electron microscope (SEM). These photos are then compared to the mask of the chip to detect additional metal or polysilicon wires. Additional metal wires and transistors can usually be detected very reliably. However, the overall process is expensive, time consuming and also destroys the chip under test. Hence, this method can only be used on a small number of chips. Also, detecting changes made to the dopant area are usually not feasible using optical reverse-engineering, especially in an automated way. We will exploit this fact to make our Trojan design invisible to this type of detection mechanism.

A different approach to test for hardware Trojans without a golden chip is functional testing of the chip. Functional testing is standard procedure in the IC design flow and to some degree will always be performed. However, detecting Trojans is different from detecting manufacturing defects. Creating efficient test cases for hardware Trojan detection is difficult since the tester does not know how the Trojan gates look like. As a result, these Trojan gates are not taken into account during the test case generation which usually tries to optimize code coverage. This leads to an inefficient functional testing procedure in contrast to functional testing at the netlist level, since in this case the Trojan gates will be part of the input to the test case algorithms.

Trojan detection mechanisms that require a golden chip have been suggested by comparing side-channel information of the golden chip and the suspected chip. The most popular method is using power side-channels for Trojan detection [7] but other
side-channels such as time [48, 80], EM and temperature have been proposed as well. Typically these detection mechanisms can only detect Trojans that are at most three to four orders of magnitude smaller than the target design [7]. Small Trojans on the other hand are likely to stay undetected. Another approach to detect Trojans is to add a specific Trojan detection circuitry into the design that can detect if the design was changed during manufacturing. For example, in [68] it was proposed to add additional gates that transform parts of the design into ring-oscillators. During testing, the frequencies of these ring-oscillators are compared with a golden chip to detect if the design was changed. These methods usually require a golden chip to determine the expected output of the detection circuitry, since circuit simulations are often not accurate enough. One big disadvantage of Trojan detection circuitry is that the circuitry itself can be subject to Trojan modifications.

For a similar reason, build-in-self-tests (BIST) that are employed in some design to automatically detect manufacturing and aging defects are of limited use when applied to Trojan detection. This is not only due to the fact that a Trojan can be inserted into the BIST itself but also because the Trojan can be designed to not trigger the BIST, since BISTs are often designed to only detect random errors.

2.4 Physical Unclonable Functions

Physical Unclonable Functions (PUF) have gained widespread attention in the research community as a new cryptographic primitive for hardware security applications. PUFs make use of the fact that two manufactured computer chips are never completely identical due to process variations. A PUF exploits these process variations to ensure that each chip has a unique behavior so that each chip can be uniquely identified. There are many applications for which PUFs can be used. Two prominent examples are the use in challenge-and-response protocols as well as for secure key generation and storage. The advantage of using a PUF to generate cryptographic keys
is that the PUF ensures that each chip will have its own unique secret without the need to program it first. Furthermore, securely storing a cryptographic key in embedded devices in a way that they are resistant to physical attacks such as probing and reverse-engineering is extremely difficult. Since with PUFs no key needs to be stored in non-volatile memory, but instead the secret is derived from physical characteristics which are hard to monitor, these attacks become extremely difficult.

In a challenge-and-response protocol, a user sends a challenge to the PUF and the PUF responds with an individual response. Each PUF instance (i.e. each individual chip) behaves differently and hence two instances will provide different responses for the same challenge. This way each chip can be uniquely identified. In the set-up phase the user chooses some random challenges and collects the responses for these challenges from the PUF. When this PUF needs to be authenticated later, the user sends a challenge he has used during the set-up phase to the PUF and compares the response with his stored response. If the received response matches the response from the set-up phase, the user can be sure that he is communicating with the same PUF instance.

PUFs can be classified into two categories: weak PUFs and strong PUFs. In a weak PUF, the number of challenges the PUFs can accept is very limited so that an attacker can try all possible challenges and store the responses. This way an attacker could easily forge the PUF by replacing the PUF with a simple memory look-up. A strong PUF on the other hand has a challenge space that is large enough so that it is computationally infeasible to try and store all possible challenges. Therefore an attacker cannot create a look-up table for a strong PUF that models the PUF correctly. Strong PUFs can be used in challenge-and-response protocols as well as for key generation. A weak PUF on the other hand cannot be used for challenge-and-response protocols but they can still be used for key generation since it is usually sufficient to generate a limited number of different keys for each chip. Note that
the terminology strong PUF and weak PUF might falsely give the impression that a strong PUF is “better” than a weak PUF. However, this terminology only defines the challenge space without judging the PUF’s performance or other security properties.

Current PUF designs face two big problems that are related: they suffer from unreliability and are prone to machine learning attacks. In an ideal case, a PUF always generates the same response for a given challenge. However, due to environmental effects and thermal noise, the response to the same challenge can vary. Therefore, error correction codes are usually needed if the PUF is used for key generation and challenge-and-response protocols need to allow a few false response bits. The second problem is that even strong PUFs can be modeled in software and the needed parameters to model a specific PUF instance can rather easily be determined using machine learning techniques if a few challenge and response pairs are known to the attacker [71]. So far, no strong PUF has been implemented that can withstand these machine learning attacks, making it currently impossible to use PUFs in a secure challenge-and-response protocol.

2.4.1 Side-Channel Resistance of Physical Unclonable Functions

Despite these problems, PUFs have gained a lot of attention. One major reason why PUFs are seen as a promising way to generate and store cryptographic keys is that tamper and side-channel resistant hardware is very expensive. Due to physical attacks such as probing and reverse-engineering, it is very difficult to store a cryptographic key in an embedded system in a secure and cost efficient way. PUFs on the other hand are believed to be very resistant against hardware attacks as any tampering with the PUF is believed to change the output behavior of the PUF and hence make extracting the key difficult. Therefore, PUFs are often seen as a promising way to generate and store cryptographic keys and there are also commercial solutions available that make use of PUFs for key generation.
The use of strong PUFs in challenge-and-response protocols has also gained a lot of attention despite their weakness towards machine learning attacks. PUFs have become a cryptographic building blocks and many protocols have been proposed including protocols with theoretical security proofs [69]. Usually three possible advantages are listed for the use of PUFs in challenge-and-response protocols: One argument for PUFs is that some PUFs have a lower area or power overhead compared to cryptographic algorithms, especially if the PUFs are compared to cryptographic algorithms that include countermeasures against hardware attacks. Secondly, the secret (key) does not need to be programmed first, ensuring that every PUF instance has a unique key even before the first power-up. This is especially useful if the PUF is used in a piracy protection scheme. And last not least, PUFs are often believed to be more secure against implementation attacks (physical attacks) such as probing and side-channel attacks. Resistance against implementation attacks is an extremely interesting property as it has been shown many times how difficult it is to protect embedded systems against these types of attacks. A very careful design and the combination of different countermeasures is needed to ensure a reasonable resistance against implementation attacks. However, this increases development costs and design overhead significantly. If PUFs would indeed be very resistant towards these implementation attacks, this would make them a very promising alternative to traditional cryptography. However, this resistance has never been thoroughly analyzed and proven.

It has already been shown that PUFs, when used for key generation, can be attacked with side-channel attacks by attacking the digital error-correction used in these systems [43, 54]. These attacks do not directly attack the used PUF itself but the digital post-processing. Nevertheless, they still indicate that using PUFs does not necessarily solve the problem of implementation attacks as other parts of the system might still be vulnerable. Recently, Merli et al. successfully attacked a ring-oscillator
(RO) PUF using an EM attack that directly targeted the PUF and not the error correction [53]. RO-PUFs only have a limited number of challenges and responses and are therefore weak PUFs. The only side-channel attacks that directly targets a strong PUF is the recent work by Delvaux et al. [25]. The authors used the large unreliability of the PUF as side-channel leakage and for the first time modeled an arbiter PUF based on this information. However, for the attack to work, the attacker needs to know the unreliability of single response bits in the presence of thermal noise. This means that the attack actually not only uses the reliability information but also directly the response bit, i.e. whether a response is biased towards 1 or 0. But if the response bits are known, traditional machine learning algorithms can be used, which are still more efficient [25]. Hence, while the paper is interesting from a theoretical point of view, it has only limited practical relevance.
CHAPTER 3
SIDE-CHANNEL BASED HARDWARE WATERMARK

In this chapter a hardware watermark is introduced that establishes a hidden communication channel in the target device using a side-channel. The advantage of this type of hardware watermark is that the watermark can be inserted at the HDL or netlist level, while the watermark can be reliably be detected if the verifier only has physical access to the suspected IC. Most other hardware watermarks can only detect watermarks at the same abstraction level as they were inserted. But in most scenarios it is unlikely that a verifier has access to higher design levels such as the netlist or the HDL code. In the following at first the two methods to implement a side-channel based watermark are discussed followed by a robustness analysis of the introduced side-channel based hardware watermark. At the end of this chapter the possibility to use the hardware watermark to protect against product piracy instead of IP theft is discussed.

3.1 Watermark Design

The main idea of the design of the side-channel hardware watermark is similar to the side-channel based hardware Trojan introduced at CHES 2009 [50]. In [50], an artificial side-channel is used to leak out secret information. In the side-channel based watermark an artificial side-channel is inserted into the IP core as well. The difference between the side-channel based hardware Trojan and the side-channel based watermark is that instead of leaking out secret information, the side-channel is engineered

1The research presented in this chapter was published in [11, 44] and was also presented at [12].
to contain a watermarking signal. Two different approaches to embed a watermarking signal into the power consumption are considered:

1. Spread spectrum based watermark

2. Input-modulated watermark

The main difference between these two approaches can be summarized as followed: For the spread spectrum watermark a single measurement with several sample points is used to detect the watermark. In contrast, the input-modulated watermark is detected by evaluating several measurements acquired at an instance of time and with different input values.

3.1.1 Spread Spectrum Based Watermark

In the spread spectrum based watermark a pseudo random number generator (PRNG) is used to generate a watermarking sequence that is leaked out by means of a low power binary amplitude modulation of the power consumption. The watermark can be revealed by correlating the correct watermarking sequence with the measured power traces. This is the same method as used in spread spectrum communication systems (also called CDMA, code division multiple access), where the transmission power of a signal is distributed over a wide frequency spectrum in a way that it can still be reliably recovered even if the signal is transmitted well below the noise floor. The same applies to our spread spectrum based watermark. The watermarking sequence can be leaked out well below the noise floor of the used side-channel and can still be reliably detected. This has two effects: First, the watermark is inherently very robust to noise and second, the watermark can be hidden below the noise floor of the power consumption and thus cannot be seen by an adversary. This hidden nature can be considered as a kind of physical encryption. The method of hiding information using spread spectrum has been used before in media watermarking schemes [24] and
other applications such as military communication. For example, the military GPS signal is encrypted and hidden in basically the same way.

3.1.1.1 Embedding the Watermark

The watermark consists of two parts: a PRNG and a leakage circuit. The PRNG needs to produce a pseudo-random bitstream, which needs to be unpredictable to allow hiding of the signal. Linear feedback shift registers (LFSR) with enough state bits to prevent brute forcing the initialization vector (IV) are well suited for this task. Nevertheless, the fact that LFSRs can easily be recovered from their known output bitstreams has to be taken into account when designing the transmission power of the watermark. An attacker must not be able to recover the transmitted bitstream from the generated leakage. This assumption holds as long as the transmitted bits are properly hidden in the noise. A way to avoid this requirement is to replace the LFSR by a secure stream cipher using a secret key. However, although stream ciphers can be implemented in hardware very efficiently, they increase the complexity of the watermark design compared to simpler PRNGs.

The circuit implementing the watermark and the PRNG should be as small as possible. This reduces the cost of the watermark while allowing to hide the design in the surrounding circuit to defend reverse-engineering attacks. Thus the design has to be balanced with respect to implementation size and PRNG strength.

The second part of our watermark, the leakage circuit, maps the PRNG output to a physical power consumption. It generates additional leakage when its input is “one” and does not generate any additional leakage when its input is “zero”. In an ASIC design, the leakage circuit can be implemented for example using big capacitances, toggling logic or pseudo-NMOS gates. In an FPGA implementation, circular shift registers can be used that are clocked according to the PRNG output. Note that the amount of generated leakage is part of the design space and can be engineered to
take any desired signal-to-noise ratio (SNR). Although the focus in this thesis is the power consumption side-channel, other side-channels such as EM leakage can be used as well.

3.1.1.2 Detecting the Watermark

The embedded watermark can be detected with similar techniques as they are used in a differential power analysis. The verifier measures a single power trace containing several clock cycles. He then compresses this power trace to a power vector with one value per clock cycle, e.g., by averaging all measurement points of each clock cycle. The verifier then simulates the bit stream generated by the PRNG using his knowledge about the implementation details. In the last detection step the verifier correlates the simulated sequence to the compressed power vector. In case he does not know the starting point of the PRNG, i.e., the correct position to align the simulated sequence to the power vector, he has to slide one (the power vector) over the other (the simulated sequence) and repeat the detection for each possible alignment. If the watermark is embedded in the examined IC, the correlation coefficient should show a prominent peak at the position of correct alignment. If the correlation coefficient does not show any significant peak, then the watermark is not embedded in the design. Using statistics to detect the watermark allows transmission with very low SNR and provides robustness to noise. The length of the simulated sequence used in the correlation step can be increased for an even more reliable detection of the watermark.
3.1.1.3 Experimental Results

To practically evaluate our proposed watermarks we used a Side-channel Attack Standard Evaluation-Board (SASEBO) [1] that is equipped with an xc2vp7 Virtex-II Pro FPGA. The power consumption leakage was measured using a LeCroy WP715Zi 1.5GHz oscilloscope at a sampling rate of 250MS/s.

We implemented a 1st order DPA resistant AES implementation and tagged it with the spread spectrum based watermark introduced above. This setup serves as our proof of concept implementation to experimentally verify our proposed watermarking scheme. As shown by Fig. 3.1 we used a 32-bit LFSR as the PRNG and connected it to a leakage circuit. The leakage circuit itself was designed from 16 look-up tables (LUTs) each configured as 16-bit circular shift registers filled with alternating ones and zeros. These registers were clocked in all clock cycles where the output of the PRNG is 1. In the first experiment we measured a long power trace covering 1000 clock cycles while the AES core was idle. The power trace was then compressed by averaging over each clock cycle and then correlated to the corresponding simulated PRNG sequence. Calculating the correlation for a window of possible alignment positions leads to the vector of correlation coefficients shown in Fig. 3.2(a). According to Fig. 3.2(b), using the leakage of around 100 clock cycles would be enough to detect the existence of the watermark in this case. In the second experiment we took a longer trace (in comparison to the first one) while the AES implementation was constantly running and processing different random inputs. This time the measured power trace covered 250,000 clock cycles, and the results of the watermark detection are shown in Fig. 3.3. Clearly adapting the number of clock cycles used for correlation can overcome the existence of the either intentional or instinctive noise and makes the detection of the watermark feasible. An implementer of the watermark has thus two handles to design the watermark detection properties: He can use longer sequences during detection or
he can increase the amplitude of the generated leakage. The later one has to be used carefully to ensure that the leakage is still hidden below the noise floor.

![Figure 3.2](image1)

**Figure 3.2.** Analysis of the spread spectrum based watermark while the AES core was idle and waiting for the next plaintext (a) using the leakage of 1000 clock cycles, (b) over the number of clock cycles

![Figure 3.3](image2)

**Figure 3.3.** Analysis of the spread spectrum based watermark while the AES core was constantly encrypting (a) using the leakage of 250 000 clock cycles, (b) over the number of clock cycles

### 3.1.2 Input-Modulated Watermark

The second approach to implement a watermark we propose in this paper uses the concept of an input-modulated hardware Trojan as introduced at CHES 2009 [50]. We call this proposal an input-modulated watermark. The idea of an input-modulated hardware Trojan is to add additional logic to the IC that results in a power consump-
tion which relies on the added logic, known input bits and some secret bits. This power consumption can then be exploited using a differential power analysis to reveal the secret bits. The main difference between the input-modulated hardware Trojan and our watermark is that the Trojan is designed to leak out secret information while the watermark is not supposed to leak out any unknown information but only its presence.

![Diagram of an input-modulated watermark](image)

**Figure 3.4.** Diagram of an input-modulated watermark that consists of an example of a combination function and a leakage circuit

### 3.1.2.1 Embedding an Input-Modulated Watermark

As shown in Fig. 3.4, the watermark logic consists of two parts, a combination function and a leakage circuit. The combination function uses some known input bits to compute one output bit. This output bit is then transmitted by means of the leakage circuit. The idea of this watermark is that we have an artificial data-dependent power consumption that is engineered by introducing the combination function. The owner of the watermark knows the implemented function and which bits were used as inputs and can use this knowledge to perform a differential power analysis. If the watermark is embedded in the IC, then this differential power analysis will be successful, while it should not be successful if no watermark or a watermark with a different combination function or different input bits is used.

To be able to implement this type of watermark some bits of the IP core need to be known by the verifier and these bits need to vary for different measurements. They
do not necessarily need to be direct inputs or outputs of the IP core but can also be
determined by internal states as long as the verifier is able to determine these bits for
a given measurement. By using internal states as “known bits” a systematic analysis
with chosen inputs can be prevented and the number of possible values used as inputs
to the combination function increases. A very easy and straightforward combination
function that was used in the proof of concept implementation of the Trojan side-
channel paper is to pairwise combine the input bits with an and conjunction and then
compute the exclusive-or sum of the outputs of all and operations.

3.1.2.2 Experimental Results

As mentioned before, the input-modulated watermark is based on the same idea
as the input-modulated side-channel hardware Trojan. In [50] practical results for a
hardware Trojan for an FPGA implementation of an AES key schedule were presented
where the input to the combination function was 8 bits of the plaintext and 8 bits
of the round key. The input-modulated watermark may use the same circuit and
changes only the inputs of the combination function. I thus omitted to present the
experiments and refer the interested reader to the experimental results already given
in [50]. For our input-modulated watermark, we would use known values for all 16
input bits of our combination function and then perform a similar correlation power
analysis. If the watermark is not embedded in the IC, then there should not be
any significant correlation between the power traces and the expected output of the
combination function.

3.1.3 Proof-Of-Ownership

The watermarks as proposed so far only allow to distinguish whether the water-
mark is embedded or not, i.e., they only provide a single bit of information. To prove
to another party that an IP core was used in a design detecting the watermark is not
sufficient, as the ownership of the watermark remains unproven. To provide proof-
of-ownership the watermarking scheme needs to be expanded to bind a watermark to a unique identity. This can be achieved by means of digital signatures. In a first step the company generates the hash value of some design ID (e.g., the part number). Then, the private key of the company is used to sign this hash value. The signature can then be transmitted by the watermarks. To allow the proposed designs of the watermarks to transmit information we again employ the initial concept of Trojan side channels. That is, for the spread-spectrum watermark, storing the information to be used for transmission in a circular shift register and XORing the MSB or LSB with the output of the PRNG to generate a specific watermarking sequence. However, an additional shift register of the size of the signature is needed for this solution, which increases the size of the watermark. Another solution to bind the digital signature to the watermark is to use parts or whole of the signature as the initial value of the PRNG. In this case the area overhead of the additional shift register can be omitted. An input-modulated watermark transmitting a signature can be designed by storing the signature in internal registers and subsequently feeding it byte-wise as inputs to the combination function in the same way the input-modulated Trojan side-channel encodes the bytes of the secret key.

The method proposed here prevents attackers from illegally claiming ownership of the used watermark, since the signature can only be generated by the owner of the private key.

3.2 Watermark Robustness

In this section we discuss three intuitive approaches to remove a side-channel based watermark: Remove or destroy the circuit implementing the watermark, increase the noise on the side-channel, or transmit an inverse watermarking signal. The later two attacks both aim at reducing the available SNR for detection of the watermark.
3.2.1 Reverse-Engineering Attack

Obviously, if the circuit implementing the side-channel watermark is destroyed or removed from the IP core, both design goals detectability and proof-of-ownership are violated. To be able to destroy or remove the watermark an attacker first needs to identify the corresponding part of the circuit. Note that the attacker does not always know whether a watermark is applied to protect a circuit and what kind of watermark is implemented. Therefore, similar to Trojan hardware, watermarks have to be implemented very small and subtle to evade identification during reverse engineering attacks. Furthermore, they should be implemented in a way to be interwoven with the surrounding functional circuit. This can further increase the difficulty to identify the watermark while impeding removal of the circuit without destroying the functionality of the surrounding circuit. Especially the input-modulated watermark can be very small, e.g., as small as around hundred gates.

The complexity of the reverse engineering attack in practice depends on the design level at which the attacker has access to the protected circuit. If the attacker has access only to the IP core at the post manufacturing level, detecting the watermark will be much more difficult than detecting it in a netlist or even in RTL sources of the design. In this case the attacker would need to first reverse engineer the hardware design to higher abstraction levels for a feasible analysis of its functionality.

Reverse engineering an entire design is difficult, complex, and thus expensive task. For watermarking purposes the goal of the watermark can be considered achieved if the efforts required to illegally use an unlicensed IP core are equivalent to the efforts necessary to reverse engineer a circuit to the level of fully understanding the design.

3.2.2 Raising the Noise

How easy a verifier is able to detect a watermark depends on the signal-to-noise ratio (SNR) of the watermarking signal. The lower the SNR, the more (or the longer)
measurements are needed to detect the watermark. Increasing the noise of the side-channel will thus reduce the SNR and therefore impedes detection. However, adding additional noise sources results in an increase of power consumption and is thus limited by practical constraints such as battery lifetimes.

Both watermarks are very robust to this kind of attacks. To detect a spread-spectrum watermark a verifier can lower the effect of the noise by averaging over multiple measurements or by increasing the number of clock cycles covered by the measured trace. For an input-modulated watermark the amount of acquired power traces used during detection is also only limited by the time required to perform the additional measurements. Since the size of the leakage circuit is a design choice of the designer, the SNR and hence the robustness to noise is part of the design space.

3.2.3 Transmission of an Inverse Watermark Signal

The idea of our third attack scenario is to hide the signal of the watermark by adding another leakage source that generates leakage in all clock cycles where the watermark itself does not generate any leakage. The idea of this attack is that this inverse signal counterbalances the original watermarking signal and results in a constant power consumption for both signals. We call this introduction of an inverse watermark signal. In theory this makes the detection of the watermarking signal impossible. We will now show that this attack can not be put into practice.

The signals of the side-channel hardware watermarks are hidden well below the noise floor and unknown to an attacker. Without the knowledge of the details of the used PRNG and its initial vector (for a spread spectrum based watermark) or the used combination function and input bits (for an input-modulated watermark) an attacker can not even compose the inverse signal. Additionally the attacker would need to know the design of the leakage generating circuit to achieve the correct amplitude for the inverse signal. We now consider that the attacker has full access
to the details of the used watermark (which is not always likely) and is able to implement an inverse copy of the watermarking circuit. This can still not avoid detection of the watermarking signal. Process variations during manufacturing as well as slight changes in internal capacitances will result in subtle differences in the power consumptions of the watermark and the inverse watermark. Also very small delays of the clock signal result in inaccurate alignments of the two signals. This makes detection of the proposed watermarking signals possible in practice even with the presence of an inverted watermarking signal.

To experimentally demonstrate this we implemented the inverse-signal transmission attack in our FPGA implementation of the spread spectrum based watermark. We added an exact copy of the watermarking circuit to our design that consists of the same leakage circuit and the same PRNG with the only exception that the output of the PRNG is inverted. The results of this attack can be seen in Fig. 3.5.

![Graphs](image)

Figure 3.5. Detection of a spread spectrum based watermark that was counter-balanced by an inverse watermarking signal. We performed the same analysis as described in 3.1.1.2 while the AES core was idle (a) using the leakage of 10,000 clock cycles, (b) over the number of clock cycles.

We have tested this configuration when the AES core was idle. It turned out that the watermark was still clearly visible with as few as 10,000 clock cycles. The correlation coefficient decreased and thus the attack led to a 10 fold increase of the required number of covered clock cycles. However, although both, the watermarking
circuit and the inverse one, use equal building blocks of the FPGA, detecting the
watermark is still possible because the power consumption of the two circuits are not
exactly inverse due to the different routing of both parts. In practice this attack is
equivalent to reducing the SNR of the watermark and cannot prevent detection of
the watermark due to the arguments given in the previous section.

3.3 Counterfeit Protection

So far we have discussed how the side-channel watermark can be used to detect
if a IP-core has been illegitimately used in an IC. But another large concern in the
IT industry is counterfeiting of products. Here we can distinguish between two types
of counterfeits: Counterfeits that are actually completely different designs (but pos-
sibly with the same functionality) but that are falsely labeled. And counterfeits that
are direct copies of the legitimate chip, either by stealing the design details (e.g. by
reverse-engineering or mask theft), by illegal overproduction of the chip by the man-
ufacturer, or because a chip that was labeled for low performance use only is illegally
relabeled for high performance.

In the case of counterfeits that are different designs, e.g., chips from a different
manufacturer that are relabeled, can easily be identified with the side-channel water-
mark without any alterations. If the watermark is not detected in the chip, the chip
under test is a counterfeit. The reason for this is that the watermarks are transmitted
below the noise level and without knowing the watermark secret the watermarks can
neither be detected nor cloned. Hence, these counterfeits will not have the correct
watermark embedded and can therefore be detected.

However, if the product theft is done by copying the design or by relabeling a
low performance chip this can not be detected without modifications to the hardware
watermark. The very idea of the watermark is that it cannot be easily removed from
a design. Hence, each copy will inevitably also have the side-channel watermark em-
bedded. To be able to protect against these types of product piracy the watermark design needs to be altered. Instead of transmitting a fixed watermark signal, the watermark design is altered to transmit a signature in a programmable part of the design. This programmable part is programmed in a step after manufacturing. For example, programmable read-only memory (PROM) based on fuses can be used to store this watermark secret. As the signature is programmed post-manufacturing, it can be generated over a device specific serial number in addition to the part number, so that each device contains a unique watermark. This way only devices programmed by the circuit designer include the correct signature and mere cloning of the semiconductor is not sufficient to build indistinguishable copies of a design. Since the signature is programmable, devices that are marked for low performance or non-commercial applications etc. can have a different signature and can therefore also be reliably detected. Previously proposed hardware watermarking schemes that are not based on side-channels cannot protect against cloning as the watermark cannot be programmed after manufacturing [42, 60, 61, 76]. In summary, a semiconductor device should include two watermarks: One that is programmed after manufacturing to protect against piracy and cloning and one that is implemented fully in hardware to detect IP theft.
CHAPTER 4
SIDE-CHANNEL BASED SOFTWARE WATERMARK

The idea behind side-channel based software watermarks is very similar to side-channel based hardware watermark. As a matter of fact, the side-channel software watermark can be seen as an adoption of the hardware watermarking techniques towards software for embedded systems. Detecting software plagiarism in embedded systems can be very challenging. In comparison to software for PC-applications the program code is often not directly accessible. Most embedded microcontrollers have a program memory read protection that prevents third parities to easily access it. Other internal memory such as RAM is usually protected as well and cannot be easily read out. This together with the fact that most programs are written in low level languages such as C or assembly make most software watermarks proposed in the literature not applicable for embedded software. The side-channel based software watermark proposed in this thesis overcomes these restriction. The watermark signal is hidden in the power consumption of the embedded microcontroller. Hence, no access to any kind of memory is needed to detect the watermark. Furthermore, the watermark can be implemented in assembly and only introduces a small overhead in terms of size and runtime, making it an attractive fit for embedded microcontrollers. Furthermore, using the EM side-channel enables an easy and contact-less way to detect the watermark.

\footnote{The research presented in this chapter was published in \cite{10} and \cite{15}.}
The basic structure of the side-channel software watermark is the same as the input-modulated watermark from Chapter 3. The watermark consists of three components:

- internal state and watermark constant
- combination function
- leakage function

and is realized by adding instructions at the assembly level to the targeted code. Figure 4.1 shows an example of a side-channel software watermark. The combination function uses some known internal state of the program and a watermark constant to compute a one-bit output. This output bit is then leaked (transmitted) via the power consumption using a leakage function. The leakage function is realized by one or several instructions whose power dissipation depends on the value of the combination function. This results in a power consumption that depends on the known internal state, the combination function and the watermark constant. To detect the watermark, the verifier can use his knowledge of the watermark design to perform a correlation power analysis (CPA). If this power analysis is successful, i.e., the watermark signal is detected in the power traces, the verifier can be sure that his watermark is embedded in the device. In many applications this will imply that a copy of the original software is present.

In the following, we will first describe the details of my proof-of-concept implementation before I explain how this watermark can be detected by means of a side-channel analysis.

4.1 Software Watermark Design

To show the feasibility of this approach, a side-channel watermark was implemented on the ATmega8 as well as on a PIC 16F687. The results for both microcon-
Figure 4.1. Example of a side-channel based software watermark that was inserted into the key expansion of an AES algorithm. The combination function as well as the leakage function is realized by adding a few additional assembly instructions into the code. The internal state needs to be changing in a predictable way. In this watermark the first two bytes of the plaintext were used as the internal state. The watermark constant is a fixed value that can be chosen by the watermark owner and forms together with the internal state the input to the combination function.

tollers are similar and in the following only the results for the Atmega8 are discussed in detail. The interested reader can find the results for the PIC 16F687 in [10].

4.1.1 Implementation

The side-channel based watermark consists of a combination function and a leakage function. In the example implementation the input to the combination function is a 16-bit internal state and a 16-bit watermark constant. The combination function computes a one-bit output which is leaked out using a leakage function. There are many ways to implement a combination function, and the function introduced here should only be seen as an example. A very small and compact combination function was chosen that only consists of four assembly instructions. The 16-bit input to the combination function is separated into two byte values $in_1$ and $in_2$. In the first step, $in_1$ and $in_2$ are each subtracted from the one-byte watermark constants, $c_1$ and $c_2$. In the next step the two one-byte results of these subtractions are multiplied with each other. The resulting two byte product $r$ from this multiplication is again separated
into two one-byte values $r_0$ and $r_1$. These two values are then multiplied with each other again.

$$r = (in_1 - c_1) \cdot (in_2 - c_2) = r_0 || r_1$$

$$res = r_0 \cdot r_1$$

The output of the combination function is the 8th least significant bit of the result $res$ of this multiplication. The corresponding assembly code can be found below.

```
subi in1,35 ;subtracts constant from in1
subi in2,202 ;subtracts constant from in2
mul in1,in2 ;multiplies in1 and in2 and
;stores the result in R0 and R1
mul R0,R1 ;multiplies R0 and R1
```

The registers $in1$ and $in2$ are used as the internal states and the integers 35 and 202 are the two watermark constants $c_1$ and $c_2$. The instruction `subi` subtracts the constant 35 from $in1$ and stores the result back in $in1$. In the ATmega8 instruction set the two-byte result of the multiply instruction `mul` is always stored in $R0$ and $R1$. The result of the combination function is the most significant bit in $R0$.

A leakage function is used to leak the output bit of the combination function. In this implementation a conditional jump is used as the leakage function. If the output bit of the combination function is 0, the two’s complement of the register $R0$ is computed, otherwise no operation is executed. Furthermore, the result of $R0$ is stored in the memory. Below is the corresponding assembly code for the leakage function:

```
SBRC R0,7 ;skip next instruction if
```
Recall that the output of the combination function is the most significant bit of R0. 

SBRC R0,7 checks the most significant bit of R0 and skips the next instruction if this bit is 0. Otherwise the neg instruction is executed, which computes the two's complement of R0. In the last step, the value of R0 is stored in the memory.

The leakage function helps the verifier to detect the watermark. The difference in the power consumption between the case that the neg instruction is executed and the case that instead a nop is executed is very big. This makes detecting the watermark using side-channel analysis straightforward.

I was also able to successfully detect the watermark without any leakage function. This is due to the fact that the power consumption of the last multiply instruction is higher if the output bit is 1 compared to 0. However, the leakage function makes detection much easier and can also protect against reverse-engineering and code-transformation attacks (see Section 4.2). The next subsection will provide more details on how the watermark detection works.

4.1.2 Watermark Verification

To detect the watermark a correlation power analysis (CPA) as introduced in Section 2.1.2 is used. The main idea of a correlation power analysis is to exploit the fact that the power consumption of a device depends on the executed algorithm as well as on the processed data. A lot of traces are measured in a CPA and then statistical methods are used to extract the wanted information. In a classical CPA setting the goal is to retrieve a secret key. In the watermark case, the verifier does not want to retrieve a secret key but wants to verify whether or not his watermark is present. To do this, the verifier first collects power traces with different inputs of
the system under test. For each trace the verifier computes the known internal state that is used as the input to the combination function (in my implementation this was the value of register \( in1 \) and \( in2 \)). The internal state used for the watermark should be a varying state that is predictable for the verifier, e.g., a state depending on the input or output. The verifier uses this internal state and the watermark constants to compute the output of the used combination function for each input value and stores these values as the correct hypothesis. He repeats this procedure \( n - 1 \) times by using different watermark constants or combination functions. At the end the verifier has \( n \) hypotheses with \( n \) different watermark constants or combination functions, where one of the hypotheses contains the correct watermark constant and combination function.

In the last step the verifier correlates the hypotheses with the measured power traces. If the watermark is embedded in the tested device, a correlation peak should be visible for the hypothesis with the correct watermark constant. This is due to the fact that the correct hypothesis is the best prediction of the power consumption during the execution of the leakage function. The result of the CPA on the example implementation can be found in Figure 4.2 and Figure 4.3. In Figure 4.3 we can see that detecting the watermark is possible with less than 100 measurements.

![Correlation vs. Time and Correlation vs. Hypotheses](image)

**Figure 4.2.** The result of the side-channel analysis plotted (a) against time and (b) with respect to different hypotheses, where hypothesis number 100 is the correct one.
Other microcontrollers will have a different power behavior and therefore the number of traces needed to detect the watermark might vary from CPU to CPU. It should be noted that a few hundred traces are easily obtained from most practical embedded systems, and it is reasonable to assume that a verifier can use much more measurements if needed. Hence, even if the signal-to-noise ratio might decrease for other microcontrollers, it is safe to assume that detection of this kind of watermark will in most cases be possible. It should also be noted that the length of the code that is being watermarked does not have an impact on the signal-to-noise ratio of the detection. The number of instructions that are executed before or after the watermark does not make any difference in this type of side-channel analysis.

4.1.3 Triggering

One important aspect of the software watermark detection is the triggering and alignment of the the power measurements. To take a power measurement, a trigger signal is needed that will indicate the oscilloscope when to start the measurement. In practice, a communication signal or the power-up signal is often used as the trigger.
signal. Other possible trigger points might be an unusual low or high power consumption, e.g., when data is written to a non-volatile memory position, the microcontroller wakes up from a sleep mode or a coprocessor is activated. Modern oscilloscopes have advanced triggering mechanisms where several trigger conditions can be used at once, e.g., a specific signal from the bus followed by an unusual high power consumption. Because these trigger points might not be close to the inserted watermark, the verifier might need to locate and align the watermark in a large power trace. Note that the designer is free to insert the watermark near a point which can serve as a convenient trigger. With some knowledge of the underlying design, it is usually possible to guess a time window in which the watermarked code is executed. Looking for power patterns that, e.g., are caused by a large number of memory look-ups or the activation of a coprocessor can also help to identify the time window where the watermark is executed. Once this time window is located, alignment mechanisms such as simple pattern matching algorithms [51] or more advanced methods such as [59, 77] are used to align each power trace with each other.

The problem of triggering and alignment of the power trace for a side-channel watermark is similar to the problem of triggering and alignment of the power traces in a real-world side-channel attack. Often in a real-world side-channel analysis the attacker has actually less knowledge of the attacked system than the verifier has for detecting the watermark. The verifier knows the code and flow of his watermarked program while in a real-world attack the attacker can often only guess how it is implemented. I would therefore like to refer to the area of real-world side-channel attacks for more details on the feasibility of triggering a measurement in practice [26, 55, 56, 63]. The alignment of power traces will be addressed in more detail in Section 4.2.2 when I describe how to overcome the insertion of random delays as one of the possible attacks on the side-channel watermark.
4.2 Robustness and Security Analysis of the Software Watermark

In the previous section I have introduced the software side-channel watermarks and showed that they can be reliably detected. However, so far the security of the watermark has not been discussed. Traditionally, the security of watermarks towards different attacks is called robustness. No software watermark has been proposed yet that is completely robust and that can withstand all known attacks [34]. For software watermarks, and especially side-channel watermarks, it is very difficult to quantify the robustness of the watermark. We do not claim that the watermark is “completely robust” or secure — given sufficient effort the side-channel based software watermark can be removed. In the following, the security model is introduced and some possible attacks against the system are described. We will provide arguments why these attacks can be non-trivial in practice. Hence, the watermark—although not impossible to remove — still represents a significant obstacle for an attacker.

In the security model of the software watermark three parties are involved: The owner of the watermark who inserted the watermark, the verifier who locates the watermark in a suspected device and an attacker who tries to remove the watermark from a software code. The attacker has only access to the assembly code of the watermarked program. The attacker does not know the design of the combination function as well as what part of the assembly code implements this combination function and which internal states or constants are being used in this combination function. This knowledge is considered the watermark secret. The verifier needs to be a trusted third party who shares the watermark secret with the owner of the watermark. A successful attack is defined as follows:

“A transformation of the watermarked software code that (1) will make it impossible for the verifier to locate the watermark with means of side-channel analysis and (2) does not change the functionality of the software program”.

50
Hence, an attacker was unsuccessful if either the verifier is still able to detect the software watermark or the resulting software code does not fulfill the intended purpose of the program any longer. I will discuss three different attack approaches to remove the watermark from the assembly code:

**Reverse-engineering attack:** In a reverse-engineering attack the attacker tries to locate the assembly instructions that implement the watermark using reverse-engineering techniques so that he can remove or alter these instructions.

**Code-transformation attacks:** In a code-transformation attack, the attacker uses automated code-transformations to change the original assembly code in a way that the resulting code is still functioning correct but the watermark detection is impossible.

**Side-channel attacks:** In a side-channel attack the attacker tries to use side-channel techniques to locate the side-channel signal in the power consumption. This gives the attacker the knowledge of the location of some of the watermark instructions (e.g., the leakage function).

In the following, each of the three attacks is discussed in more detail.

### 4.2.1 Reverse-Engineering Attack

If the attacker can reverse-engineer the entire code and identify the purpose of each instruction and function, the attacker also knows which instructions are not directly needed by the program and which are therefore possible watermark instructions. However, complete reverse-engineering of the assembly code can be very difficult and time consuming, especially in larger programs. Furthermore, complete reverse-engineering might be more expensive than actually implementing it, making product piracy not cost effective if reverse-engineering is needed. An attacker can try to locate the watermark without reverse-engineering the entire code. For example, the attacker could use techniques such as data-flow diagrams to detect suspicious code segments
which he can then investigate further. The complexity of such attacks depends on the attackers reverse-engineer skills as well as on the way the watermark is embedded in the code.

I believe that due to the small size of the watermarks, locating the watermarks with methods of reverse-engineering can be very expensive for the attacker. Especially in larger designs, which are usually more attractive for software theft, this can be very difficult. Another attractive property of the side-channel watermarks is that they are hidden in the power consumption of the system. This means that an attacker cannot tell whether or not a watermark is present in the code. So even if she locates and removes one or several side-channel watermarks from a code, she cannot be sure if there are not still more watermarks present in the code. Considering the small size of only 5-10 assembly instructions for some watermarks, adding multiple watermarks is still very economical. This may discourage attackers from stealing the code as reverse-engineering the entire code is necessary to ensure that all watermarks have been removed.

4.2.2 Code-Transformation Attacks

In an automated code-transformation attack, a software is used to change the program code without changing the semantically correct execution of the program. Examples for code-transformations are recompiling the code, reordering of instructions and obfuscation techniques such as replacing one instruction with one or more instructions that have the same result. Code-transformations can be a very powerful attack tool for disabling software watermarks as has been shown in [34], where all tested static software watermarks for Java bytecodes have been successfully removed with standard obfuscation tools.

Let us first consider the impact of reordering of instructions and the insertion of dummy instructions on the side-channel watermark. If these methods are used by the
attacker, they can have the effect that the leakage function is executed in a different clock cycle compared to the original code. For the detection this means that the correlation peak will be at a different clock cycle. However, the correlation peak will be as visible as without the reordering as inserting a static delay does not decrease the signal-to-noise ratio. Therefore, simple reordering and the insertion of dummy instructions cannot prevent the verifier from detecting the watermark.

However, if the attacker does not add a static but a random delay this will have a negative impact on the watermark detection. Random delays have the effect that the measurement traces are not aligned with each other, i.e., the clock cycle where the leakage function is executed varies from measurement to measurement. Unaligned traces hamper side-channel analysis but the detection can still be successful if enough traces are aligned with each other [51]. It is not always easy to insert efficient random delays into the code, e.g., a source of randomness is needed and simply measuring the execution time of a program might give indication of the random delay introduced. Furthermore, the verifier can use alignment methods to detect such misalignment and remove the random delays. By using these alignment methods the verifier has a good chance to counteract the random delays, especially if the delays are inserted several clock cycles before the leakage function. Due to the fact that the attacker does not know the location of the leakage function this is very likely. Otherwise the attacker would need to insert a lot of random delays which will hurt the performance.

To show the power of alignment techniques to counteract random delays I changed the experiment by inserting random delays. In my first approach I added my own random delays by using a timer interrupt that would pseudo-randomly trigger every 1-128 clock cycles. I used an S-Box to generate the pseudo-random numbers and an externally generated 8-bit random number as its initialization for each measurement. These random interrupts did not provide much of an obstacle and with a simple pattern matching algorithm [51] I was able to detect the watermark. The result
of the CPA is shown in Figure 4.4(a). To make the experiment more credible I also implemented the random delay based side-channel countermeasure presented at [22] into the watermarked AES. This countermeasure, called *improved floating mean*, inserts random delays at fixed positions but with a varying length. The initial state of the used PRNG is an externally generated 64-bit number\(^2\). I again used a pattern-matching alignment algorithm and peak extraction before performing a CPA. The result of this analysis is shown in Figure 4.4(b). The correlation coefficient decreased for this analysis compared to the original watermarked AES implementation from around 0.85 to 0.5 but this correlation value is still very large. These results aim to show that it is not simple to insert random delays to defeat the watermark detection. With more improved alignment methods (e.g. [59, 77]) better results could probably be achieved. Furthermore, Strobel *et al.* [75] demonstrated how to efficiently remove random delays from power traces.

By replacing instructions, an attacker might change the power profile of a code. For example, instead of using the decrement instruction \texttt{DEC} to decrease a register value, the subtract with constant instruction \texttt{SUBI} could be used. These instructions have a different power profile. However, even if the attacker can change the power profile of the code significantly, this does not impact the CPA. The power consumption of the clock cycles before or after the watermark do not have impact on the CPA correlation. As long as there is a difference in power consumption according to the output of the combination function this difference can be used to detect the watermark using a CPA. For example, just the transmission of the output of the combination function over the internal bus usually leaks enough data-dependent power

\footnote{I used the parameters provided in [22] for my implementation of *improved floating mean* with three dummy rounds before the encryption and inserted the watermark in the main AES encryption function. 29 random delays, each varying between 24-536 clock cycles in steps of two, are executed before the first execution of the watermark.}
Figure 4.4. Figure showing a CPA with 1,000 measurements to detect the watermark in two implementations with random-delay countermeasures. In both figures peak extraction and a pattern based alignment method were used. In (a) random delays are introduced by pseudo-randomly triggering a timer interrupt every 1-128 clock cycles. In (b) the side-channel countermeasure improved floating mean was added to the watermarked AES. In both cases the watermark is clearly detectable.

A code-transformation that removes the output of the combination function on the other hand would be successful. But every code-transformation algorithm needs to make sure that the resulting code does not change the semantically correct execution of the program. Therefore, it needs to be ensured that for the compiler or code-transformation algorithm the watermark value is considered needed. This can be done by storing the output or using it in some other way. In this case any code-transformations attacks will be unsuccessful as removing the watermark value would destroy the semantical correct execution of the program from the view of a compiler.

Adding additional side-channel watermarks to a program can be seen as a code-transformation as well. A side-channel watermark should not change the state of the

---

This assumes that the power consumption of the internal bus is correlated with the Hamming weight of the transmitted bits, which is usually the case for microcontrollers[51].
program that is being watermarked to ensure that the watermark does not cause software failures. Hence, additional watermarks will not change the combination function of a previously inserted watermark. They might only introduce some either static or data-dependent delays. For this reason, it is possible to add multiple watermarks into a design without interference problems.

4.2.3 Side-Channel Attacks

If the attacker can successfully detect the watermark using a side-channel analysis, the attacker also gains the knowledge of the exact clock cycles where watermark instructions (e.g., the leakage functions) are executed. In this case the attacker only needs to remove or alter these instructions to make the watermark detection impossible. Therefore, the watermark should only be detectable by the legitimate verifier who possesses the watermark secret.

The attacker can try to discover the watermark secret by performing a brute-force side-channel analysis in which he tries every possible watermark secret. But the attacker faces two problems with this approach: the big search space of possible watermarks and false positives. The size of the search space of possible watermark secrets depends strongly on the application, the size of the watermark, and the architecture of the microcontroller. The application that is being watermarked determines how many internal states can be used as inputs to the combination function and the size of the watermark determines how many operations the combination function performs. Finally, the number of available instructions and functions that can be used for the watermark also influences the search space.

In the following, I give a rough estimation of a possible search space for the example application of an AES-128 encryption on the 8-bit ATmega8 microcontroller. The AES encryption program has two 16-byte inputs, the plaintext and the key, and one 16-byte output, the ciphertext. Let us assume that the designer of the watermark
can use the 16-byte input for his internal state of the watermark. For simplicity I assume that the combination function consists of 10 basic instructions using the internal states and two 8-bit watermark constants as inputs. Furthermore I assume that only the six ATmega8 instructions addition, subtraction, AND, OR, exclusive-OR, and multiplication can be used. Using these parameters the lower bound of possible different combination functions is roughly $2^{75}$.

Besides the large search space an attacker has also to face the problem of false positives. If an attacker tries $2^{75}$ different watermark secrets it is likely that the attacker will see some correlation peaks that are not due to the actual watermark. One reason for a correlation peak might simply be noise as statistically some hypotheses will generate greater correlation peaks than others. Such peaks are called ghost peaks in the literature. These correlation peaks should be smaller than the actual correlation peak due to the watermark if enough traces are used. However, if the attacker has not discovered the watermark yet he does not know how high the correlation peak of the watermark is supposed to be and might therefore falsely suspect wrong parts of the design to be the watermark. The second reason why a false positive might appear is that some part of the actual program that is being watermarked might be linearly related to a possible watermark. In a brute-force approach all possible operations on the internal states are tested. Therefore, it is more than likely that one or several of the tested combination functions are identical or linearly related to parts of the actual program. In this case, correlation peaks appear that will indicate a possible watermark at a location where there is no watermark embedded.

To summarize, detecting the watermark using side-channel analysis can be quite complicated for the attacker. Using small (and possible multiple) watermarks will increase the problem of false positives while the search space becomes too big in practice if larger watermark secrets (more operations and/or more possible internal
states) are used. In my opinion, it seems that using a reverse-engineering or code-transformation approach is more promising to remove the watermark in practice.

4.3 Proof-of-Ownership

The watermark discussed in the previous section only transmits one bit of information: either the watermark is present or not. This is very helpful to detect whether or not your code was used in an embedded system. However, it is not possible for the verifier to prove towards a third party that he is the legitimate owner of the watermark. This is due to the fact that the watermark itself does not contain any information about the party who inserted the watermark. Therefore, everyone could claim to be the owner of the watermark once he detects a watermark in a system. In this section, I will show how we can expand the watermark idea to provide proof-of-ownership as well.

The idea to establish proof-of-ownership is to modify the side-channel watermark in a way that the watermark transmits a digital signature that can uniquely identify the owner of the watermark. This approach is similar to the proof-of-ownership concept for hardware watermarks in Section 3.1.3. One attractive property of the side-channel watermark is that the watermark is hidden in the noise of the power consumption of the system. Without the knowledge of the watermark secret, the presence of the watermark cannot be detected. So we have already established a hidden communication channel. In Figure 4.2(a) we can observe a positive correlation peak while the leakage circuit is being executed. Recall that the leakage circuit is designed in such a way that the power consumption is higher when the output of the combination function is ‘1’. If we change the leakage function in a way that the power consumption is higher when the output bit is ‘0’ instead of ‘1’, then the correlation will be inverted. That means that we would not see a positive, but a negative correlation peak. We can use this property to transmit information. If the bit we want to
transmit is ‘0’, we invert the output bit of the combination function. If it is ‘1’, we do not invert the output bit. By doing so, we know that ‘1’ is being transmitted when we see a positive correlation peak and ‘0’ when we see a negative correlation peak. We can use this method to transmit data one bit at a time. I tested this kind of watermark by using the same combination function as discussed in Section 4 but exchanged the leakage function. I stored an 80-bit signature that I wanted to transmit with my watermark in the program memory. I then load the signature, one byte at a time, from the program memory and subsequently add the output bit of the combination function to the bit we want to transmit. The resulting bit is then leaked out using a conditional jump, just as it has been done in Section 4.1.1.

The same detection method as explained in Section 4.1.2 is used to detect the watermark and to read out the transmitted signature. Figure 4.5 shows the result of this correlation based power analysis. The positive and negative correlation peaks over time represent the transmitted signature. The resulting watermark is still quite small. In the example implementation the watermark consists of only 15 assembly instruction for the leakage function and only 4 instructions for the combination function. I also used 80 bits of the program memory to store the digital signature. If storing the signature in the program memory is too suspicious, it is also possible to implement the leakage function without a load instruction by using constants. This might increase the code-size a bit, but it is still possible to program the leakage function with around 30 instructions for a 80 bit signature on an 8-bit microcontroller. In a 16-bit or 32-bit architecture, smaller code sizes can be achieved.

4.4 Detecting Software Theft Without a Watermark

The introduced side-channel based software watermark can be used to reliably detect software theft in embedded systems. The lightweight nature of the watermark makes it easy and economical to insert such watermarks. However, what if no water-
Figure 4.5. Side-channel watermark that transmits an ID. Positive correlation peaks indicate that a ‘1’ is being transmitted, negative correlation peaks indicate a ‘0’. In this Figure we can see how the hexadecimal string “E926CFFD” is being transmitted.

mark has been added to the design before a suspected software theft? Even in this case it is still possible to detected software theft in embedded systems using power side-channels [15]. At the example of the Atmega 8 microcontroller it was shown that it is possible to determine the Hamming weight of the executed machine code from the power consumption of the device. This enables a verifier to compare his software code with the code run on a device without having access to the program memory. To test if your software is running on the device, a verifier collects the power consumption of the device under tests and reads out the Hamming weight of the executed instructions. The verifier can then compare this Hamming weight string with the corresponding Hamming weights of his software code. However, the measurements can be faulty so that some Hamming weights might be wrong and interrupts and conditional branches can change the execution flow of the software. Hence, it

\[I\] would like to note that my co-author Daehyun Strobel deserves the credit for this technique and it was only included in this thesis for sake of completeness.
is likely that the two strings will not match completely, even if the same software is used. Dot-plots can be used to overcome this weakness and detect the software theft in the presence of interrupts and branches. For details on this technique I would like to refer to [15]. The disadvantage of this technique is that it is not as robust against code-transformations as the watermark solution and that it only works with microcontrollers that directly leak out the Hamming weight of the executed instructions. The software watermark on the other hand is much more robust to noise due to its differential nature and does not rely on such a fine-grain power model.

Hence, the side-channel based software watermark is more reliable in detecting software theft. But if no watermark is available, the technique presented in [15] based on string-matches can be a promising alternative for detecting software thefts.
CHAPTER 5
SIDE-CHANNEL BASED HARDWARE TROJANS

The previous chapters showed how intentional side-channels can be used to build efficient watermarks for protecting software and hardware designs. However, intentional side-channels can also be used maliciously. This chapter discusses one possible malicious application of intentional side-channels: hardware Trojans. Hardware Trojans are malicious modifications to a hardware design that alter the functionality or reliability of a design or leak out secret information. Hardware Trojans can be activated with a specific event, such as a specific input sequence, or at a specific time. But some hardware Trojans can also be active all the time. A good hardware Trojan should remain undetected at least until it has performed the intended action. Otherwise the Trojan-infected hardware will be discarded before the attacker has gained any advantage.

The hidden nature of intentional side-channels make them an attractive tool to build hardware Trojans: Intentional side-channels are well hidden in the noise and can only be recovered by their implementer. Furthermore, intentional side-channels can be built with only a few gates and are hard to detect using reverse-engineering. In the following a novel way to build layout level hardware Trojans is introduced. These so-called dopant Trojans are realized without adding any additional transistors. Instead, only existing gates are modified below the polysilicon level, making them difficult to detect. The “usefulness” of these Trojans are demonstrated in Section 5.1.2 where it is demonstrated how the security of an otherwise secure random

\[ \text{The research presented in this chapter was published in [14].} \]
number generator (RNG) is compromised by inserting this Trojan into the digital post-processing of this RNG. In Section 5.2 I then demonstrate that the same technique can be used to efficiently establish a Trojan side-channel to leak out the secret key of an otherwise side-channel resistant encryption algorithm. The side-channel is realized without adding a single transistor and has zero overhead in terms of area.

In Section 5.3 results of side-channel based hardware Trojans that are inserted at the gate level as opposed to the layout level are summarized. Interestingly, the use of side-channels in regards to hardware Trojans are not restricted to transmit information from within a IC to the outside. Intentional side-channels can also be used to activate a Trojan by transmitting a trigger signal over a side-channel towards the Trojan circuitry. These active side-channel Trojans are discussed in Section 5.3.2.

5.1 Dopant Trojans

In this section we focus on Trojans which are inserted by modifying the layout of a design. Focusing on the layout level represents a major difference to most other work in the hardware Trojan research. This work helps to shed light on the question of how hard or easy it is for a foundry to insert a Trojan if they do not have access to higher abstraction levels than the mask. Furthermore, by focusing on the layout level we can set a new lower boundary of how much overhead can be expected by a hardware Trojan in practice. Such information is essential for research and work in the Trojan detection area. As it turns out, an intentional side-channel can be introduced into a design without even adding a single transistor. The intentional side-channel is created by only modifying the layout masks below the polysilicon level without touching the polysilicon level or any metal layer. This reduces the overhead of the Trojan in terms of gate-count and area to zero. This new type of Trojan, called dopant Trojan, is also close to invisible to optical inspection, making it extremely difficult to detect.
5.1.1 Design of Dopant Trojans

In this section an efficient way to design hardware Trojans without changing any metal or polysilicon layer of the target design is introduced. The main idea of the proposed Trojan is teh following: A gate of the original design is modified by applying a different dopant polarity to specific parts of the gate’s active area. These modifications change the behavior of the target gate in a predictable way and are very similar to the technique used for code-obfuscation in some commercial designs [40]. In the following I will explain the dopant modification method at the example of a simple inverter. In this example the inverter is modified in a way that it always outputs $V_{DD}$ regardles of its input. I would like to note that the proposed techniques are sufficiently general to be applied to other types of gates in a similar way.

An inverter consists of a pmos and an nmos transistor whose drain contacts are connected via a metal layer as shown in Figure 5.1(a). In the upper part of Figure 5.1(a) shows a pmos transistor which consists of an nwell and a polysilicon wire separating the positively doped source and drain region within the active area. To create an inverter Trojan that constantly outputs $V_{DD}$, the positively doped p-dopant mask of this pmos transistor is exchanged with the negatively doped n-dopant mask. Doping an active area within an n-well with n-dopant basically creates a connection to the n-well. N-wells are usually always connected to $V_{DD}$ in a CMOS design. Since the n-dopant is applied to the entire active area of the pmos transistor, including the metal contacts, a direct connection from these contacts to the n-well is created. The upper part of Figure 5.1(b) shows the resulting pmos transistor Trojan. The source contact, which is connected to $V_{DD}$, has been transformed into an n-well tap, creating an additional connection from the n-well to $V_{DD}$. The drain contact is also connected to the n-well and thereby to $V_{DD}$. Hence, we have created a constant connection between $V_{DD}$ and the drain contact without modifying the metal, polysilicon, n-well or active area.
In the second step the connection between the nmos transistor’s drain contact and GND is constantly disabled. This is achieved by applying p-dopant to the source contact of the nmos transistor while leaving the drain contact untouched. Applying p-dopant to the source contact of the nmos transistor transforms it into a well tap again and cuts of any connection between the source contact and the negatively doped source area of the nmos transistor. Therefore, the nmos transistor is no longer connected to GND regardless of its gate input. The resulting Trojan inverter can be seen in Figure 5.1(b). The metal, ploysilicon, active and well layers are identical with the original inverter in Figure 5.1(a), but the Trojan gate always outputs $V_{DD}$ regardless of its input. Note that by simply switching the roles of the nmos and pmos transistors the inverter can also easily be modified to constantly output $GND$ regardless of its input.

Figure 5.1. An unmodified inverter gate (a) and a Trojan inverter gate with a constant output of $V_{DD}$ (b).
Besides fixing the output of transistors to specific values, it is also possible to change the strength of transistors in a similar way. The strength of a transistor in CMOS is defined by its width. Usually the entire active area of a transistor is doped and therefore the width of a transistor is defined by the active area. However, by decreasing the area that is doped positively in a p-mos transistor, it is possible to reduce also the effective width of the transistor. Hence, to decrease the strength of a transistor it is sufficient to apply p-dopant to an area smaller than the active area of the transistor. Since the active area and the p-dopant and n-dopant layers use different masks, this can easily be done without changing the manufacturing process.

One of the major advantages of the proposed dopant Trojans is that they cannot be detected using optical reverse-engineering. Detecting changes done to metal or polysilicon layers, and to some degree also to active areas, can be relatively easily detected using scanning electron microscopes. Changes to the dopant layer, on the contrary, are almost invisible to these techniques. This properties is succesfully commercially deployed for code-obfuscation [40]. This fact demonstrate that our newly proposed dopant Trojans are extremely stealthy as well as practically feasible.

How these modifications can be used in a larger design to build useful and stealthy hardware Trojans is discussed using two case studies as examples, a Trojan inserted into the cryptographically secure post-processing of Intel’s RNG design as it is used in the Ivy Bridge processors, and an SBox implementation of a side-channel resistant logic style.

5.1.2 Case study: A Dopant-Trojan for a Secure RNG Design

In this section we apply the concepts of our dopant Trojans to a meaningful, high-profile target to demonstrate the danger and practicability of the proposed Trojans. Our first target is a design based on Intel’s new cryptographically secure RNG. Most prominently, it is used in the Ivy Bridge processors but will most likely be used in
other designs in the future. We chose this target because of its potential for real-world impact and because there is detailed information available about the design and especially the way it is tested [33, 4, 79].

The cryptographically secure RNG generates unpredictable 128-bit random numbers. The security has been verified by an independent security company [33] and is NIST SP800-90, FIPS 140-2, and ANSI X9.82 compliant. We will modify the digital post-processing of the design at the sub-transistor level to compromise the security of keys generated with this RNG. Our Trojan is capable of reducing the security of the produced random number from 128 bits to \( n \) bits, where \( n \) can be chosen. Despite these changes, the modified Trojan RNG passes not only the Built-In-Self-Test (BIST) but also generates random numbers that pass the NIST test suite for random numbers.

In the following section we first summarize the design of Intel’s RNG and then discuss our malicious modifications.

### 5.1.2.1 Intel’s Secure TRNG Design

Like most modern RNGs, Intel’s RNG design consists of an entropy source (ES) and digital post-processing. Figure 5.2 provides an overview of the RNG design. The design also features a Built-In-Self-Test (BIST) unit that checks, at each power up, the correct functioning of the entropy source and the digital post-processing.

The netropy source is a metastable circuit based on two cross coupled inverters with adaptive feedback. The digital post-processing consists of a Online Health Test (OHT) unit and a cryptographically secure Deterministic Random Bit Generator (DRBG). The OHT monitors the random numbers from the entropy source to ensure that the random numbers have a minimum entropy.

The Deterministic Random Bit Generator itself consists of two parts, a conditioner and a rate matcher. The conditioner is used to compute new seeds for the rate
matcher. Based on the current state, the rate matcher computes 128-bit random numbers. Reseeding is done whenever the conditioner has collected enough random numbers from the entropy source, or if at most 512 128-bit random numbers have been generated by the rate matcher. The conditioner as well as the rate-matcher are based on AES.

![Diagram](image)

**Figure 5.2.** Overview of Intel’s RNG design. An Entropy Source (TRNG) generates truly random numbers whose entropy is monitored by the Online Health Test (OHT). The random numbers are then fed to a digital random bit generator (DRBG) consisting of a Conditioner and a Rate Matcher. The Conditioner is used to periodically reseed the Rate Matcher which provides the output RnRand of the RNG. The correct functioning of the RNG is checked at each power up using the Build-In Self Test (BIST). The Trojan is inserted into the 256 bit state of the Rate Matcher.

The rate matcher generates the 128-bit output $r$ of the RNG and takes the seed $(s, t)$ generated by the conditioner unit as input. The rate matcher has two internal state registers: a 128-bit register $K$ and a 128-bit register $c$. During normal operation, the rate matcher generates 128 random bits $r$ and updates the state registers in the following way $(r, c, K) = \text{Generate}(c, K)$:

1. $c = c + 1$, $r = AES_K(c)$

2. $c = c + 1$, $x = AES_K(c)$
3. $c = c + 1, y = AES_K(c)$

4. $K = K \oplus x$

5. $c = c \oplus y$

Whenever the conditioner has a new seed, consisting of the 128-bit values $s$ and $t$, available the internal states $c$ and $K$ are reseeded using the $(c,K)=$Reseed$(s,t,c,K)$ function:

1. $c = c + 1, x = AES_K(c)$

2. $c = c + 1, y = AES_K(c)$

3. $K = K \oplus x \oplus s,$

4. $c = c \oplus y \oplus t$

Under low load, the rate matcher reseeds after each output of $r$. Under heavy load, the rate matcher generates several random numbers $r$ before it reseeds, up to a maximum of 512. However, even under heavy load, the rate matcher should reseed long before reaching its maximum of 512 [33].

5.1.2.2 Dopant-Trojan for Intel’s DRBG

A 128-bit random number $r$ generated by the rate matcher is the result of an AES encryption with an unknown 128-bit random input $c$ and an unknown, random key $K$. The attacker has a chance of $1/2^{128}$ to correctly guess a random number resulting in an attack complexity of 128-bits. The goal of our Trojan is to reduce the attack complexity to $n$ bits, while being as stealthy as possible. This is achieved by cleverly applying our dopant-based Trojan idea described in Section 5.1.1 to internal flip-flops used in the rate matcher. In the first step we modify the internal flip-flops that store $K$ in a way that $K$ is set to a constant. In the second step the flip-flops storing $c$
are modified in the same way, but \( n \) flip-flops of \( c \) are not manipulated. Hence, only \((128 - n)\) flip-flops of \( c \) are set to a constant value. This has the effect that a 128-bit random number \( r \) depends only on \( n \) random bits and \( 128 + (128 - n) \) constant bits known to the Trojan designer. The owner of the Trojan can therefore predict a 128-bit random number \( r \) with a probability of \( 1/2^n \). This effectively reduces the attack complexity from 128-bit down to \( n \) bits. On the other hand, for an evaluator who does not know the Trojan constants, \( r \) looks random and legitimate since AES generates outputs with very good random properties, even if the inputs only differ in a few bits.

Our Trojan can be implemented by only modifying the flip-flops storing \( c \) and \( K \), while all other parts of the target design remain untouched. Two different Trojan flip-flops are needed: one which sets the flip-flop output to a constant ‘1’ and one which outputs a constant ‘0’ regardless of the inputs. The DFFR_X1 flip-flop of the used Nangate Open Cell library [39] has two outputs, \( Q \) and its inverse \( QN \). To implement our Trojan, the drain contact of the p-MOS transistor that generates signal \( Q \) is shortened to \( V_{DD} \) by applying n-dopant above the drain contact, as explained in Section 5.1.1. Simultaneously, the source contact of the n-MOS transistor for signal \( Q \) is disabled by applying p-dopant to the source contact. Hence, the output signal \( Q \) generates a constant output of \( V_{DD} \) regardless of its input. The inverse output \( QN \) is modified in the same way, only that this time the drain contact of the n-MOS transistor is shortened to GND and the source contact of the p-MOS transistor is disabled. This leads to a constant output of ‘0’ for \( QN \). The same modifications are used to generate a flip-flop Trojan to constantly provide an output of \( Q=’0’ \) and \( QN=’1’ \) by switching the roles of the n-MOS and p-MOS transistors. Note that only four of the 32 transistors of the DFFR_X1 flip-flop are modified as can be seen in Figure 5.3. But 28 transistors on the other hand stay untouched and therefore will
still switch according to the input. This results in a smaller but still similar power consumption for a Trojan flip-flop compared to a Trojan-free flip-flop.

![Trojan area](image)

**Figure 5.3.** Layout of the Trojan DFFR_X1 gate. The gate is only modified in the highlighted area by changing the dopant mask. The resulting Trojan gate has an output of $Q = V_{DD}$ and $QN = GND$.

### 5.1.2.3 Defeating Functional Testing and Statistical Tests

It is a standard procedure to test each produced chip for manufacturing defects. In addition to these tests, the produced RNGs will also be tested against a range of statistical tests in order to be NIST SP800-90 and FIPS 140-2 compliance. Furthermore, to be compliant with FIPS 140-2, the RNG needs to be tested at each power-up to ensure that no aging effects have damaged the RNG. For this purpose Intel’s RNG design includes a Built-In-Self-Test unit that checks the correct functioning of the RNG in two steps after each power-up. In the first step, the entropy source is disabled and replaced by a 32-bit LFSR that produces a known stream of pseudo-random bits. The BIST uses this pseudo-random bit stream to verify the correct functioning of the OHT and feeds this bitstream to the conditioner and rate matcher. A 32-bit CRC checksum of the $4 \times 128$-bit output buffer that stores the last
four outputs $r_1, \ldots, r_4$ of the rate matcher is computed. This 32-bit CRC checksum is compared against a hard-coded value to verify the correct functioning of the conditioner and rate matcher. If the checksum matches, the RNG has passed the first part of the BIST. In the second part of the BIST the conditioner, rate matcher and output buffer are reset and the entropy source is connected again. The OHT tests the entropy of the entropy source and simultaneously seeds the conditioner and rate matcher. If the OHT signals the BIST that the entropy of the entropy source is high enough, the BIST is passed and the RNG can generate random numbers.

In [4] it is stated that “This BIST logic avoids the need for conventional on-chip test mechanisms (e.g., scan and JTAG) that could undermine the security of the DRNG.” This fact is also mentioned in an Intel presentation in which it is argued that for security reasons the RNG circuitry should be free of scan chains and test ports [79]. Therefore, to prevent physical attacks, only the BIST should be used to detect manufacturing defects. From an attacker’s point of view, this means that a hardware Trojan that passes the BIST will also pass functional testing. Although Intel’s BIST is very good at detecting manufacturing and aging defects, it turns out that it cannot prevent our dopant Trojans. One simple approach to overcome the BIST would be to add a dopant Trojan into the BIST itself to constantly disable the error flag. However, it could be very suspicious if the BIST never reports any manufacturing defects.

To pass the BIST, the Trojan rate matcher needs to generate outputs $r'_1, \ldots, r'_4$ during the BIST that have the same 32-bit CRC checksum as the correct outputs $r_1, \ldots, r_4$. Since the input to the rate matcher during the BIST is known, the Trojan designer can compute the expected 32-bit CRC checksum. He then only needs to find a suitable value for the Trojan constants $c[1 : 128]$ and $K[1 : 128 - n]$, which generate the correct CRC checksum for the inputs provided during the BIST. Since the chance that two outputs have the same 32-bit CRC is $1/2^{32}$, the attacker only needs $2^{32}/2$
tries on average to find values for $c$ and $K$ that result in the expected 32-bit CRC. This can easily be done by simulation. By cleverly choosing $c$ and $K$ the Trojan now passes the BIST, while the BIST will still detect manufacturing and aging defects and therefore raises no suspicion.

Since the Trojan RNG has an entropy of $n$ bits and uses a very good digital post-processing, namely AES, the Trojan easily passes the NIST random number test suite if $n$ is chosen sufficiently high by the attacker. We tested the Trojan for $n = 32$ with the NIST random number test suite and it passed for all tests. The higher the value $n$ that the attacker chooses, the harder it will be for an evaluator to detect that the random numbers have been compromised.

Detecting this Trojan using optical reverse engineering is extremely difficult since only the dopant masks of a few transistors have been modified. As discussed, detecting modifications in the dopant mask is extremely difficult in a large design, especially since only a small portion of a limited number of gates were modified. Since optical reverse-engineering is not feasible and our Trojan passes functional testing, a verifier cannot distinguish a Trojan design from a Trojan-free design. This also means that the verifier is not able to reliably verify a golden chip. But without such a verified golden chip, most post-manufacturing Trojan detection mechanisms do not work.

5.2 Side-Channel based Dopant-Trojans

With the second case study we want to emphasize the flexibility of the dopant Trojan. Instead of modifying the logic behavior of a design, the dopant Trojan is used to establish a hidden side-channel to leak out secret keys. We prove this concept by inserting a hidden side-channel into an AES SBox implemented in a side-channel resistant logic style.

We chose the side-channel resistant logic style iMDPL for our target implementation despite the fact that it has some known weaknesses, namely imbalanced routing,
that can enable some side-channel attacks [58]. Our target iMDPL SBox is reasonably secure and we would like to stress that the focus of this work is hardware Trojans and not side-channel resistant logic styles. Our point here is that our Trojan modifications do not reduce the side-channel resistance against common side-channel attacks while enabling the Trojan owner to recover the secret key. In the following section a brief introduction of iMDPL is given and then the dopant based side-channel Trojan is explained.

5.2.1 Target iMDPL Design

The improved Masked Dual Rail Logic (iMDPL) was introduced in [65] as an improvement of the Masked Dual-Rail Logic (MDPL) [66]. There are three main ideas incorporated in iMDPL:

1. Dual-Rail: for every signal \( a \), both the true and the complementary signal (indicated with \( \bar{a} \)) are computed. Therefore the same number of 1’s and 0’s are computed regardless of the input. This prevents attacks based on the Hamming weight.

2. Precharge phase: Between two clock cycles, there is always a precharge phase in which all iMDPL gates (besides registers which have to be treated differently) are set to 0. This prevents attacks based on the Hamming distance.

3. Mask bit: Due to imbalances in routing inverse signals and process variations, the power consumption of a signal \( a \) might differ from that of its inverse signal \( \bar{a} \) which can lead to side-channel attacks. In iMDPL a random mask bit is used to randomly choose between \( a \) and \( \bar{a} \) to mask the power consumption.

In an iMDPL gate, every input and output bit as well as its inverse is masked with a mask bit \( m \). An iMDPL-AND gate performing the operation \( q = a \& b \) has six inputs: The masked input values \( a_m = a \oplus m \), \( \bar{a}_m = a \oplus \bar{m} \), \( b_m = b \oplus m \), \( \bar{b}_m = b \oplus \bar{m} \).
and the mask bit $m$ and its inverse $\bar{m}$. The two outputs of an iMDPL-AND are $q_m = q \oplus m$ and $\bar{q}_m = q \oplus \bar{m}$.

The schematic of an iMDPL-AND gate is shown in Figure 5.4. It consists of a detection stage, an SR-latch stage and two majority gates with complementary inputs. If one input of a 3-input majority gate is set to 0, the majority gate behaves like an AND gate. If one input is set to 1, the majority gate behaves like an OR gate. For the mask bit $m = 0$, the lower Majority gate with the inputs $a_m$, $b_m$ and $m$ computes $q = q_m = a \& b$ and the upper majority gate computes $\bar{q} = \bar{q}_m = \bar{a} \mid \bar{b}$. For the mask bit $m = 1$ on the other hand the lower majority gate computes $\bar{q} = q_m = \bar{a} \mid \bar{b}$ and the upper majority gate computes $q = \bar{q}_m = a \& b$. Hence, the current mask bit decides which inputs and outputs are the correct ones and which the inverse. It is also possible to create an iMDPL-OR and iMDPL-NOR gate using the same structure by switching the outputs and/or inputs. In iMDPL all combinational logic is build using these four basic operations (AND, NAND, OR and NOR). The detection and SR-latch stage was introduced in iMDPL to prevent the early propagation effect and glitches by making sure that all inputs are in a complementary stage before evaluating. A more detailed description of iMDPL can be found in [65].

As in the previous sections, the 45nm Nangate Open Cell library was used for our implementation of an area optimized Canright [20] AES SBox in iMDPL. Since the target library does not have a 3-input majority gate, we used a six input AND-OR-INVERTER (AOI) gate configured as a 3-input not-majority gate together with an inverter to build the majority gate$^2$.

$^2$We would like to note that the layout of a majority gate is very similar to an AOI gate and we verified that the Trojan also works with a standard majority gate.
Figure 5.4. Schematic of an iMDPL-AND gate consisting of two Majority gates, a detection logic and an SR-latch stage[65].

5.2.2 iMDPL Dopant Trojan

To insert a Trojan into the iMDPL SBox implementation, we replace two AOI gates from a single iMDPL gate with Trojan AOI gates that create a predictable, data-dependent power consumption independent from the mask bit. Modifying only single gates makes inserting the Trojan into the design after place & route very simple, since we do not need to worry about any additional routing or find empty space in the design. Figure 5.2.2 shows the schematic of the used AOI gate configured as a 3-input not-majority gate. Two changes are made to this not-majority gate to create a large data-dependent power consumption. First, the two topmost p-MOS transistors are removed by shorting their output contacts to VDD. Secondly, the strength of the remaining p-MOS transistors is decreased by decreasing their effective width. These changes are depicted on the right side of Figure 5.2.2.

The Trojan not-majority gate behaves like the Trojan-free gate except for the input pattern $A = 0$, $B = 1$, and $C = 1$. In the unmodified not-majority gate the pull-up network is inactive and the pull-down network is active, resulting in an output value of 0. However, in the Trojan gate the pull-up as well as the pull-down
network are both active for this input pattern. Due to the reduced size of the p-MOS transistors, the pull-up network is much weaker than the pull-down network and the resulting output voltage is therefore still close to 0. In a sense we have turned the not-majority gate into a pseudo-n-MOS gate for this input pattern. Hence, the output values of both the Trojan-free and Trojan gate are the same, but there is a large power consumption in the Trojan gate for this input pattern due to the connection between \( GND \) and \( V_{DD} \). For all other inputs only the pull-up or pull-down network is active for the Trojan gate as well as the Trojan-free gate.

If the two not-majority gates of the iMDPL gate are exchanged with this Trojan gate, a high power consumption is generated whenever one of the two AOI gates has the input \( A = 0, B = 1, \) and \( C = 1 \). In our configuration this is the case if \( a_m = 0, \) \( b_m = 1, \) \( m = 1 \) or if \( \bar{a}_m = 0, \) \( \bar{b}_m = 1, \) \( \bar{m} = 1 \) which turns out to be the case for \( a = 1, b = 0 \) regardless of the value of \( m \). Hence, the Trojan iMDPL gate has a data-dependent power consumption that is independent of the mask bit \( m \).

**Figure 5.5.** Schematic of the Trojan-free and Trojan AOI222_\( X1 \) gate configured as a 3-input not-majority gate.
We used the technique of dopant Trojans described in Section 5.1.1 to realize our Trojan AOI gate. The modifications were done using Cadence Virtuoso Layout editor and are shown in Figure 5.6(b). The Trojan gate passed the DRC check and we used Calibre PEX in Virtuoso to do the netlist and parasitic extraction. The Trojan and Trojan-free gate were simulated in HSpice. The propagation delay, rise and fall time of a Trojan iMDPL gate are very similar to the Trojan-free iMDPL implementation. This makes it possible to place our Trojan gates even in the critical path without creating timing violations. The additional power consumption when the Trojan activates depends on the used clock frequency, since the majority of power consumption of the Trojan is static current due to the connection between $V_{DD}$ and $GND$. Even at a very high frequency such as 10 GHZ, the Trojan gate consume roughly twice as much power when the Trojan activates compared to the Trojan-free counterpart.

![Figure 5.6](image)

**Figure 5.6.** On the left (a) the layout of the unmodified AOI222_X1 gate and on the right (b) the Trojan AOI222_X1 gate is shown. In the Trojan gate the p-MOS transistors in the upper left active area have been shorted with the n-well by replacing the p-implant with n-implant. The strength of the remaining p-MOS transistors in the upper right active area have been reduced by decreasing the p-implant in this area.

To insert our Trojan iMDPL gate in the layout of the target SBox implementation after place & route we need to identify an iMDPL gate that serves as a suitable Trojan
location and replace the AOI gates of this target iMDPL gate with the Trojan AOI gate. Finding a suitable location does not require a detailed knowledge of the target SBox. In fact, the right location can be identified using simulation. The individual iMDPL gates can easily be identified by searching for AOI gates connected with inverse inputs. In the first step, we simulated the SBox for all 512 possible inputs (for each mask there are 256 different inputs) and stored the inputs and outputs for the tested AOI gates. Then, a matlab script was used to test the performance of possible Trojan target locations. We chose a target location that (1) had a small correlation with the Trojan power model for all false key guesses to make it easy for the owner of the Trojan to use it and (2) a location which did not increase the vulnerability against the considered side-channel attacks. We tested (2) by performing the considered side-channel attacks on hypothetical power traces based on the Trojan power model. Once we located a good Trojan location we simply replaced the corresponding AOI gates with the Trojan AOI gate.

5.2.3 Trojan Effectiveness Evaluation

To verify the correct functioning of the Trojan iMDPL gate and to show that the Trojan gates do not violate timing requirements even if the Trojan gate is placed in the critical path, transistor-level simulations were performed. These simulations of a Trojan and a Trojan-free iMDPL gate were performed using HSpice and the parasitic extraction was done by Calibre PEX. Table 5.1 shows the delay, rise and fall times of the Trojan-free and the Trojan iMDPL gate for a load capacitance of 5.4fF, the equivalent of the gate capacitance of two iMDPL gates. The Trojan-free and the Trojan gate have very similar timing characteristics. In most cases the Trojan gate is even faster than the Trojan-free gate. Therefore, it is possible to exchange an iMDPL gate with a Trojan iMDPL gate even if it is in a time-critical path.
Table 5.1. Performance of the Trojan-free and Trojan iMDPL-AND gate for different input patterns and a load capacitance of 5.4fF. A and B depict the unmasked inputs to the iMDPL-AND gate and M the mask bit. The column “0 → 1” shows the propagation delay of the evaluation phase and “1 → 0” the propagation delay of the precharge phase in picoseconds. “Risetime” represents the risetime of either $Y_m$ or $\overline{Y}_m$ during the evaluation phase.

For the simulation of the entire SBox after place & route Synopsys Nanosim was used. Nanosim is not as precise as HSpice but the used simulation configuration\(^3\) is still very precise and does take internal and routing capacitances into account. The needed interconnect parasitics were extracted using Cadence Encounter and Calibre PEX was used again to extract the transistor level parasitics of the Trojan and Trojan free AOI gate. The transistor level parasitics for the other gates were taken from the NangateOpenCell library. The side-channel analysis was performed using Matlab scripts.

To verify the correct functioning of our Trojan we performed a side-channel attack with the Trojan power model using the Trojan Sbox implementation and the Trojan-free implementation on simulated power traces. Figure 5.7(a) shows the result of the attack on the Trojan SBox and Figure 5.7(b) shows the result of performing

\(^3\) Simulations were performed with Synopsys Nanosim using the following configuration: sim=4, model=4, net=4, set powernet default mode=5, set sim ires 1pA, set print ires 1pA and set sim leak ires=1fA.
the same attack on the Trojan-free implementation. The correct key can clearly be distinguished for the Trojan SBox with a correlation close to 1. It is also interesting to note that the Trojan generates leakage current compared to switching current. Hence, one can make power measurements after most switching activity has occurred and use integration to increase the signal-to-noise ratio. This makes using the Trojan easy in a practical setting. As expected, the Trojan power model does not reveal the key in the Trojan-free implementation, which shows that the side-channel was indeed produced by the added Trojan.

![Correlation vs. Time](image)

(a) Trojan design  
(b) Trojan-free design

**Figure 5.7.** 1-Bit CPA on (a) the Trojan design and (b) the Trojan-free design using the Trojan power model with the evaluation phase starting at 0ns and the precharge phase starting at 15ns. The correct key is shown in black and the false keys are shown in gray. The correlation for the correct key in the Trojan design goes up to 0.9971.

We then compared the side-channel resistance of the Trojan implementation with the Trojan-free implementation. Covering all possible side-channel attacks out of the scope of this paper. We therefore only considered some of the most common side-channel attacks, namely 1- and 8-bit CPA [17] and MIA [28]. We found a small vulnerability in the Trojan-free design, which is in line with the results from [58]. However, the Trojan did not increase this weakness and the Trojan design is as side-channel resistant as the Trojan-free design against the considered side-channel attacks.
In the following a detailed analysis of the side-channel resistance of the Trojan and Trojan-free design is provided.

### 5.2.4 Side-Channel Analysis of the Trojan and Trojan-Free Design

The target iMDPL SBox implementation suffers to a certain degree from the general problem of iMDPL, pointed out by Moradi et al. [58], namely that the different masks have different routing and timing characteristics and therefore a different power profile. During the precharge phase a Correlation-Power Analysis (CPA) [17] using an 8-bit Hamming weight (HW) model is feasible as can be seen in Figure 5.8(a). However, the simulations are noise free, meaning there is no algorithmic noise from other SBoxes and other parts of the AES algorithm such as MixedColumns. Furthermore, no measurement noise nor thermal noise is present as well as no filtering effects due to sense resistors and capacitances of the global power supply as in a real measurement. In a real-world scenario the measured correlation would therefore be much smaller and the attack not trivial. For all other time instances outside the shown 14.8ns to 16ns window the 8-bit CPA was unsuccessful.

![Figure 5.8](image)

**Figure 5.8.** 8-Bit HW CPA attack on (a) the Trojan-free design and (b) the Trojan design with the precharge phase starting at 15ns. The correct key, highlighted in black, can be distinguished. However, the correlation coefficient of the attack for both the Trojan and Trojan-free is the same.
Figure 5.9. Figure (1) shows the result of a MIA attack with a 1024 bin histogram method and an 8-bit HW distinguisher for the Trojan design. The correct key, highlighted in black, never reaches the maximum for any time period and therefore the attack is unsuccessful. On the right the results of a 1-bit CPA on the first bit of the SBox output of the Trojan design is shown. Again the correct key, highlighted in black, can not be distinguished from false keys at any time instance.

But the insertion of the Trojan does not have any impact on the attack as can be seen in Figure 5.8(b). The correlation during the point of attack does not increase but stays the same. For all other time instances the correct key cannot be recovered using the 8-Bit CPA and therefore the Trojan does not decrease the security against this attack.

1-bit CPA attacks on the output of the Trojan design were unsuccessful for all 8 bits. As an example, Figure 5.9(b) shows the result of the 1-bit CPA attack using the least significant SBox bit. For no point in time does the correct key reach a maximum. Interestingly, Mutual Information Analysis (MIA) [28] was unsuccessful even at the time instance where the CPA was successful for both the Trojan as well as the Trojan-free design. A 1024 bin histogram method with an 8-bit HW distinguisher was used for the first MIA on the Trojan design. The result of this attack can be seen in Figure 5.8(b). The correct key cannot be distinguished from false keys since the correct key never reaches a maximum mutual information value. We assume that — likely by happenstance of the routing algorithm — the leakage during precharge must

83
behave very close to the Hamming weight power model and therefore the CPA is more successful than MIA. Before using an area-optimized Canright AES SBox we tested our iMDPL design flow using a table look-up based AES SBox which resulted in a much larger design. For this design, MIA actually performed better than CPA in the 8-bit HW model which suggests that it depends on the design and routing algorithm if MIA attacks have better results than CPA attacks. We also tried to increase the bin size to 20000 but the MIA remained unsuccessful.

Further, MIA attacks with a distinguisher based on 4 or 6 bits of the SBox output were performed. These attacks were unsuccessful for both the Trojan as well as Trojan-free design. It is important to note that the Trojan was placed in a well chosen location so that the Trojan power model does not have a direct relation to the SBox output. This explains why the Trojan was not detected by MIA attacks since the output of the SBox is used as an input to the distinguisher function. The well chosen position of the Trojan is also the main reason why the 1- and 8-bit CPA attack did not reveal the Trojan. We specifically chose the Trojan to not be triggered by these power models.

Since the Trojan did not reduce the resistance to 8-Bit HW CPA attacks compared to the Trojan-free implementations and all other attacks were unsuccessful on the Trojan design, the introduced Trojan does not reduce the side-channel resistance against the considered side-channel attacks. Therefore, to an evaluator testing these side-channel attacks, the Trojan design appears to be side-channel resistant. The owner of the Trojan on the other hand can use the secret Trojan power model to successfully attack the design. But we would like to note that this does not mean that the Trojan is undetectable using other side-channel attacks. However, the analysis shows that if the attacker knows the methods the design will be tested again (e.g. because it is listed in a standard) he can specifically design the Trojan so that these attacks are unsuccessful.
The side-channel analysis showed that we have successfully established a hidden side-channel that can leak out secret keys very reliably while not decreasing the side-channel resistance against the most common side-channel attacks. Hence, the newly introduced Trojan side-channel can only be used by the owner of the Trojan who knows the secret Trojan power model. Since we did not change the logic behavior of any gate, functional testing cannot detect the Trojan. As discussed in Section 5.1.1, detecting this Trojan using optical inspection is very challenging since we only modified the dopant masks. Without being able to detect the Trojan using functional testing or optical inspection, an attacker cannot distinguish a Trojan chip from a Trojan-free chip. Hence, an evaluator cannot verify a golden chip and therefore methods that rely on a golden chip have only limited use in detecting the Trojan. This shows that detecting a dopant-based side-channel Trojan would be challenging in practice using known methods.

5.3 Side-Channel based Hardware Trojans at the Gate Level

The layout level hardware Trojan introduced in the previous sections are especially well suited to be inserted by a malicious factory. However, hardware Trojans can also be inserted at other stages in the design process. Hardware Trojans could, for example, be inserted into 3rd part IP-cores. In this case, the Trojans will likely not be realized at the layout level but at the gate level, since many IP-cores are sold as a netlist or as HDL code. In the following some results on how side-channels Trojans can be inserted at the gate level are summarized.
5.3.1 Passive Side-channel Trojans

The techniques discussed in Chapter 3 can directly be applied to hardware Trojans [44, 50]. The main difference is that a hardware Trojan does not transmit a signature but some kind of secret information, e.g., a key. As a matter of fact, the side-channel based hardware watermark design was mainly derived from the Trojan side-channels presented in [50]. Except for small modifications, the design of a side-channel watermark and a side-channel hardware Trojan is the same. Both, the input modulated as well as the spread spectrum approach, can be used to implement hardware Trojans. Figure 5.10 [50] shows the principle of the Trojan side-channel. The main idea is to add additional gates to the design which establish a hidden communication channel using an intentional side-channel to leak out a secret key. This is achieved by adding additional gates that do not change the logical output of the design, but only modifies the power consumption. The number of additional gates needed for this side-channel Trojan can be very small. In the proof-of-concept implementation in [50] the input-modulated Trojan, which could leak out parts of the round keys during an AES encryption, only consisted of roughly 20 gates.

One nice property of side-channel Trojans is that they do not need an activation mechanism. Because they do not change the output of the design, they can always be active. Hence, Trojan detection approaches that are based on detecting the activation mechanism, such as functional testing, will not detect side-channel based Trojans. Most other Trojan detection mechanisms proposed in the literature require a golden model. A golden model is a reference chip that does not have any hardware Trojans embedded. However, it is not always guaranteed that such a golden model exists. If the Trojan is inserted at the design stage, every chip has the Trojan embedded

---

4 The subsection “Passive side-channel Trojans” describes content developed mainly by my colleagues Wayne Burleson, Tim Güneysu, Markus Kasper, Lang Lin, Oliver Mischke, Amir Moradi and Christof Paar, and and was included for sake of completeness.
Figure 5.10. Principle of the Trojan side-channel [50]. A Trojan side-channel $c$ that leaks out the secret key $K$ is inserted into the power consumption of the device. Only the attacker knowing the Trojan secret can recover the key $K$ from $c$. An evaluator on the other hand cannot distinguish $c$ from noise.

and no Trojan-free circuit exits. To detect the Trojan an evaluator can try use use general side-channel analysis to detect if the design leaks out a secret. However, such a side-channel analysis extremely difficult, since the Trojan side-channel does not follow common power models. If the Trojan side-channel is well designed, such analysis will not be able to detect the Trojan side-channel. Another approach to detect this type of Trojan is to use some kind of data-flow analysis to detect that the additional gates are not needed for the correct functioning of the IP core. However, clever implementing the Trojan can counteract this.

How powerful such a side-channel based hardware Trojan could be in practice was demonstrated in [44]. The side-channel Trojan is used to leak out the mask of a masked AES encryption. Without knowing the mask values, the design is side-channel resistant. However, since the Trojan established a hidden communication channel to covertly leak out the mask value for each encryption, the design can be attacked by the Trojan owner. What makes this Trojan so powerful is the fact that detecting this hidden communication channel is close to impossible. If the key would be leaked out directly, this might be possibly be detected using sophisticated side-channel attacks such as collision attacks or fixed-vs-random tests [29]. However, since not the key but
the secret and randomly generated mask bits are leaked instead, these approaches do not work.

Inserting additional gates into a design after place and route is not a straightforward task. There are typically unused spaces in the design but adding additional gates also requires additional routing. How difficult it is to add these gate level hardware Trojans at the manufacturing stage is therefore still an open research problem.

Furthermore, even the insertion of only a few gates by a foundry at the mask level can be detected reliably using optical reverse-engineering. Using optical reverse-engineering, the verifier can compare photos of a decaped chip at different layers with the corresponding mask. Small differences between the mask and the die photos might not be recognized due to dirt on the lenses etc. However, additional metal wires and gates can easily be detectable using an automated recognition algorithm, even if there are millions of transistors in a design.

In summary, gate level side-channel Trojans are especially well suited to be inserted into IP-cores before manufacturing. Detecting these Trojans are very difficult as no activation mechanism is needed and functional testing will not reveal them. But if gate level Trojans are inserted at during manufacturing, optical reverse-engineering can reveal the additional gates and especially the additional routing. Note that optical reverse-engineering is a destructive technique and quite time and labor intensive. Hence, it can be assumed that only a few chips will be examined with this technique.

5.3.2 Active Side-channel Trojans

In the previous sections we have always used intentional side-channels to transmit data from the IC to the outside world. However, side-channels can also be used to transmit data from the outside to the IC. Such an active side-channel can for example be used to activate a hardware Trojan as has been demonstrated at the NYU Poly embedded system challenges. For example, during the 2010 challenge our
team implemented a timing based Trojan that would activate not on specific inputs, but on a specific delay pattern between inputs [13]. Hence, for this Trojan time was used as a side-channel to transmit the trigger to the IC. But many more side-channels besides time can be used as triggers. For example, we proposed to activate the Trojan by altering the power supply of the Trojan in a specific pattern. The idea is to have a ring-oscillator whose frequency is directly dependent on the supply voltage. If the supply voltage is lowered the frequency of the ring oscillator decreases. This voltage sensor can then be used as the input to a Trojan activation circuitry that monitors the time the chip is in a high and low voltage state. When a specific pattern of high and low input voltages is applied to the circuitry the Trojan is activated. While we only proposed this Trojan but not implemented it in the NYU Poly challenge, the team from Iowa State University implemented a similar idea and used heat to activate the Trojan [70]. The idea to transmit and receive data using the heat side-channel has also been proposed in [18]. Please note that these intentional side-channel are realized using only digital logic and hence no analog logic is needed.
CHAPTER 6
SIDE-CHANNEL ATTACKS ON DELAY-BASED PUFs

In this chapter we will take a closer look at the side-channel resistance of strong PUFs against passive and attacks. In particular, two different classes of implementation attacks are considered, power side-channel analysis and fault attacks. As our target we chose the Arbiter PUF, as it is the most widely discussed strong PUF in the literature. We show that it is possible to attack the Arbiter PUF even in scenarios where machine learning attacks are infeasible, i.e. when the attacker does not directly have access to the individual response bits. Hence, our results raise strong doubt whether the assumption that delay based PUFs are less prone to implementation attacks than traditional cryptographic algorithms holds true. It also raises questions about the usefulness of delay-based PUFs in general. Since they loose much of their

Most proposed weak PUF designs are either memory based or based on ring-oscillators. The Arbiter PUF is the most promising strong PUF design that has gained a lot of research attention. A detailed description of how Arbiter PUFs work and how they can be modeled is presented in the next Section. In Section 6.1.3 a secure PUF designed based on an Arbiter PUF is introduced under the assumption that the PUF achieves a reliability close to 100%. This secure design will be used as the target design for the active and passive side-channel attacks in Section 6.2.2 and 6.3.2 respectively.
6.1 Target PUF Design

6.1.1 Arbiter PUF

The basic idea of the Arbiter PUF is to apply a race signal to two identical paths and determine which of the two paths is faster. The two paths have an identical layout so that the delay difference $\Delta D$ between the two signals mainly depends on process variations. This dependency on process variations ensures that each chip will have a unique delay behavior. The Arbiter PUF gets a challenge as its input which defines the exact paths the race signal takes. Figure 6.1 shows the schematic of an Arbiter PUF. It consists of a top and bottom signal that is fed through delay stages. Each individual delay stage consists of two 2-bit multiplexers (MUX) that have identical layouts and that both get the bottom and top signals as inputs. If the challenge bit for the current stage is '1' the multiplexers switch the top and bottom signals, otherwise the two signals are not switched. Each individual transistor in the multiplexers has a slightly different delay characteristic due to process variations and hence the delay difference between the top and bottom signal are different for a '1' and a '0'. This way the race signal can take many different paths: an $n$-stage Arbiter PUF has $2^n$ different paths the race signals can take. However, challenges that only differ in a few bits have a very similar behavior so that an Arbiter PUF does not necessarily have $2^n$ unique challenges. An Arbiter at the end of the PUF determines which of the two signals was faster. The Arbiter consists of two cross-coupled AND gates which form a latch and will have an output of '1' if the top signal arrives first and '0' if the bottom signal is the first to arrive. The Arbiter can have a slight bias so that the PUF result might be slightly biased towards '0' or '1'.

6.1.2 Modeling an Arbiter PUF

Arbiter PUFs can easily be modeled in software if the delays that are added by each individual stages are known. Each stage has four delay values: the delay for
the top and bottom signal for challenge bit '1' and for challenge bit '0'. Since we are actually not interested in the total delay but only in the delay difference, we can reduce these four values to two values per stage \( i \), the delay difference \( \delta_{1,i} \) between the top and bottom signal for challenge bit '1' and the delay difference \( \delta_{0,i} \) for the challenge bit '0'. The delay difference is positive if the top signal is faster and negative if the bottom signal is faster. The total delay difference \( \Delta D \) for a given challenge \( \vec{C} = c_1, \ldots, c_n \) can be computed easily by adding up the individual delays for each stage. If the challenge bit is '1' the wires are switched and the top signal becomes the bottom signal and vice versa. The switching of the wires can be modeled by simply multiplying the current delay difference with minus one. The time difference \( \Delta D_i \) between the top and bottom signal after stage \( i \) can therefore easily be expressed recursively with the following equation:

\[
\Delta D_i = \Delta D_{i-1} \ast (1 - 2 \ast c_i) + \delta_{c_i,i}
\]

The final time difference between the two signals is simply time difference \( \Delta D_n \) after the last stage \( n \) and the response bit \( r \) is defined by:

\[
r = \begin{cases} 
1 & \text{if } \Delta D_n > 0 \\
0 & \text{if } \Delta D_n < 0 
\end{cases}
\]
This way the PUF can be modeled with $2n$ delay values. However, a more efficient approach to model an n-stage Arbiter PUF that only requires $n + 1$ parameters is used in practice. A PUF instance is described by the delay vector $\vec{w} = (w_1, ..., w_{n+1})$ with:

$$w_1 = \delta_{0,1} - \delta_{1,1}$$
$$w_i = \delta_{0,i-1} + \delta_{1,i-1} + \delta_{0,i} - \delta_{1,i} \text{ for } 2 \leq i \leq n$$
$$w_{n+1} = \delta_{0,n} + \delta_{1,n}$$

The delay difference $\Delta D_n$ at the end of the Arbiter is the result of the scalar multiplication of the transposed delay vector $\vec{w}$ with a feature vector $\vec{\Phi}$ that is derived from the challenges:

$$\Delta D_n = \vec{w}^T \vec{\Phi}$$

The feature vector $\vec{\Phi}$ is derived from the challenge vector $\vec{c}$ as follows:

$$\Phi_i = \prod_{l=i}^{n} (1 - 2c_l) \text{ for } 1 \leq i \leq n$$
$$\Phi_{n+1} = 1$$

Modeling a PUF in this way can significantly decreases the simulation time and also reduces the parameters that need to be known to $n + 1$. It was shown in the past how these parameters can be computed (or approximated) easily using different machine learning techniques. In practice, only a few hundred challenge and response pairs are needed to model an Arbiter PUF with a predication rate very close to the reliability of the attacked PUF [38]. To make machine learning attacks more difficult, some designs try to add a non-linear component to the Arbiter PUF e.g. by using feed-forwards or by XORing the responses of several Arbiter PUFs. Although these designs make machine learning attacks more difficult, they still do not provide a big challenge from an attackers perspective. It has been shown that it is still fairly easy to
achieve a prediction accuracy of above 98% percent using machine learning techniques such as Evolution Strategies (ES) or Linear Regression (LR)[71].

6.1.3 Controlled PUF Design

As mentioned, unreliability and their weakness against machine learning attacks are currently the main drawback of Arbiter PUFs. However, these two problems are closely related: If non-linearity is added to an Arbiter PUF by means of a combination function such as an XOR or by a feed-forward, the reliability decreases. This decrease in reliability limits the number of non-linear additions that can be added to the Arbiter PUF, since otherwise the PUF becomes too unreliable to be useful. Furthermore, in some cases the accuracy of the machine learning attacks decreases slower than the reliability, so that adding additional non-linearity can be counter productive from a security perspective.

Since current PUF designs can easily be attacked using machine learning if the attacker has access to challenge and response pairs, so called Controlled PUFs have been proposed to prevent these attacks. The term controlled PUF was introduced by Gassend et al. in [27] and the main idea is to add additional circuitry that prevents an attacker to apply arbitrary challenges and, most importantly, hides the individual response bits from the attacker. The main idea of most controlled PUFs is to instead of applying the challenges to the PUF directly, a master challenge is sent to the chip. From this master challenge, a challenge generator generates $n$ individual challenges that are applied to the PUF. The $n$ individual response bits of this PUF are not directly returned as outputs but instead are first applied to a cryptographically secure one-way function (e.g. a hash function). Figure 6.2 shows an example implementation of a Controlled PUF design and which is used as a case study in the remainder of this chapter. By never revealing the individual response bits to the outside, machine
Figure 6.2. The controlled PUF design. An 80-bit master challenge is applied to
the controlled PUF from which 80 individual sub-challenges are derived using the
challenge generator. These 80 sub-challenges are applied to the 128-bit Arbiter PUF
and the 80 PUF responses are stored in a shift register. This 80-bit string is hashed
using a cryptographically secure one-way function and the resulting 64-bit hash value
is provided as the final response of the controlled PUF.

learning attacks are not feasible any longer\(^1\).

Of course, the proposed design adds a non negligible overhead to the PUF design,
as a challenge generator and one-way function is needed. However, there exist several
secure lightweight encryption algorithms that could be used for this purpose. The
challenge generator generates \(n\) sub-challenges for a single master challenge. There
are two main requirements for the challenge generator:

1. A master challenge should not generate sub-challenges that are similar i.e. sub-
challenges that have large sequences of bits that are equal between these sub-
challenges.

2. Two master challenges should not generate sub-challenges that similar with each
other.

\(^1\)This assumes that \(n\) is chosen sufficiently large and the PUF has enough entropy that brute-force
attacks are not possible.
A simple LFSR for example would not be a good challenge generator, since a master challenge of all '0's would generate individual challenges that also only consist of all '0's. In this case all response bits will be the same as all individual challenges are identical. But if none or only a few response bits are different it is not difficult to reverse the one-way function since the entropy and therefore the computation complexity is reduced greatly.

An example of a secure challenge generator would be a block cipher in counter mode with the key as the master challenge. By definition, changing a single input or key bit in a block cipher changes each output bit with a probability of 50%. This makes it computationally infeasible for an attacker to find master challenges that break requirement 1) or 2). A block cipher \( c = enc_k(p) \) can also be used as a secure one-way function \( y = f(x) \) by encrypting a constant \( C \) using \( x \) as the key: \( y = enc_x(C) \). By definition, it should be computationally infeasible to compute the key of the block cipher even if the attacker has access to plaintext-ciphertext pairs. Hence, for a given \( y \) it is computationally infeasible to compute \( x \). The fact that the same design can be used as a challenge generator and as a one-way function greatly reduces the introduced area overhead of a controlled PUF. Many lightweight block ciphers exist such as PRESENT [16] or NSA’s lightweight block ciphers [9] that could be used for this purpose. However, the results presented in this paper are sufficiently general that other approaches such as using a secure hash function as the challenge generator and secure one-way function can be used instead.

It is also important to note that the physical security requirements for the challenge generator and the one-way function are much lighter than for the case that they are used with a traditional secret key that needs to be protected. The challenge generator only processes known values, hence it does not need to be resistant against information leakage. The one-way function also has reduced physical security requirements since it does not process a constant secret. For each master challenge,
the input to the one-way function is different and unpredictable. Hence, differential attacks such as DPA and CPA are not feasible. Therefore only implementation attacks that can reveal information with a single input, e.g. a simple power analysis, need to be considered. But defending against these types of side-channel attacks is much easier than defending against differential side-channel attacks and most hardware block-ciphers are secure against simple power analysis. Hence, to attack such a system using implementation attacks, the attacker would need to directly attack the Arbiter PUF and not the digital post-processing².

In the following we will assume that a design as depicted in Figure 6.2 is used with a block cipher such as PRESENT as the challenge generator and one-way functions as well as an 80-bit shift register to store the 80 individual response bits. However, since our attacks will actually directly target the 128-bit Arbiter (or the registers storing the Arbiter response) and not the digital pre- or post-processing the results are also valid for other designs that use an Arbiter PUF with known challenges.

### 6.2 Power Side-Channel Attack on Arbiter PUFs

In this section we will take a closer look at the resistance of Arbiter PUFs towards passive power side-channel attacks. We assume that the attacker does not have access to the challenge and response pairs because a controlled PUF design as described in section 6.1.3 is used. The question is if the attacker can gain enough information out of power side-channel measurements of the PUF to successfully model it. In the following we will first take a closer look at the power consumption of the Arbiter PUF and then show how this information can be used to attack controlled PUFs in a combined machine learning and power side-channel attack.

---

²As mentioned, this assumes implementation attacks cannot reveal secret information from a single trace. Some probing and fault attacks could therefore in theory still be used to attack the hash function instead of the PUF.
6.2.1 Power Consumption of Arbiter PUFs

To evaluate information leakage of an Arbiter PUF in the power consumption we performed some simulations to test the power behavior of an 128-bit Arbiter PUF. Figure 6.3 shows the power consumption of the tested 128-Bit Arbiter PUF in 45nm technology. The PUF is the same design as described in Section 6.1.3 with one addition: the result of the PUF is stored in a register. It is a reasonable assumption to assume that the response bit is stored in a register, since the PUF response needs to be processed further. There are two points of interest in the power traces. One point of interest is when the Arbiter at the end of the PUF determines whether the output is 1 or 0 and the second point of interest is when the result is stored in the register. In this simulation the register was reset before the PUF execution. Figure 6.4 shows the correlation of 2000 power traces with the correct response bits as well as the correlation with response bits with a prediction accuracy between 50% to 90%. There is a strong correlation of the correct response bits and the power traces at two time instances. At around 1350 ns when the PUF evaluates the response the power traces show a correlation of -0.32 and when the response bit is stored in the flip flop at 1600 ns the correlation is close to 1.

The reason why the correlation is negative at 1350 ns is because the power consumption is actually higher if the response bit is 0 instead of 1 in this design due to the implementation details of the Arbiter PUF. The fact that a correlation of close to 1 is achieved during the storing of the result in the register is not very surprising since it is well known that a register has a large power consumption when the output value changes. For prediction accuracies below 100% the correlation coefficient decreases linearly. Hence, the correlation coefficient directly relates to the model accuracy and it is possible to distinguish PUF models with a high accuracy from PUF models with a low accuracy using the correlation coefficient.
Since we only use a single register in this simulation, it is actually possible to directly read out the response from the power trace. In practice, more switching activity will likely occur from other sources such as state machines and challenge generators. This introduces algorithmic noise to the power traces in addition to measurement and environmental noise so that directly reading out the response bits is likely impossible in practice.

![Power Trace Graph](image)

**Figure 6.3.** Two power traces of an 128-bit Arbiter PUF for two different challenges, one challenge with a response of 1 and one with a response of 0.

The data-dependent power consumption during the storing of the result is much larger than the power consumption during the evaluation due to the fact that the power consumption of a register is the highest when it is switching. However, in practice the power consumption during the evaluation of the Arbiter might also be an interesting attack point since it evaluates very late in the clock cycle and hence there might be less algorithmic noise compared to the register which evaluates at the rising edge of the clock cycle. In our controlled PUF design the response of the PUF is stored in an 80-bit shift register. The assumption that the result of the arbiter is stored in a $n$-bit shift-register is very reasonable if a design is used in which the
Figure 6.4. The correlation of responses with different accuracies with 2000 simulated power traces of a 128-bit Arbiter PUF.

PUF is called $n$ times before the result is processed. Using a shift-register is the most common and most efficient approach to store data in this case. The power consumption of a shift register follows a Hamming distance model: If two consecutive bits are different, then the stored values in the registers change and generate a large power consumption. On the other hand, if two consecutive bits are the same then the values do not change and hence consume only a small amount of power. The power consumption of the shift-register is therefore directly proportional to the Hamming distance between the current state of the shift register with the previous state.

During the evaluation of the Arbiter on the other hand the power consumption is independent of the previous response bit. This is due to the fact that the PUF is always set to the same state before the race signal is applied. Hence, the data-dependent power consumption during the end of the evaluation phase of the Arbiter PUF depends on a Hamming weight model. We have actually omitted a third time instance in which the power consumption directly depends on the response bit. When the Arbiter PUF is reset to the initial state to prepare for the next race signal there
is again a data-dependent power consumption comparable to the power consumption during evaluation.

It should also be noted that after the power-up the shift register will contain all zeros. In this specific case the power consumption of storing the first response bit in the register is also independent of previous responses and follows the Hamming weight model. An attacker could power off and then power-on the device between each measurement if he wanted to use the Hamming weight power model together with the power consumption of the register. In our attack, the Hamming distance power model on the shift register greatly outperformed the Hamming weight power model so that this seems to be unnecessary or counter-productive for most attacks.

In practice, the side-channel measurements will contain noise from various sources. The different noise sources are typically physical noise, measurement noise, model matching noise, and algorithmic noise. Physical noise sums up noise sources such as supply noise, thermal noise, and temperature differences. Measurement noise is added by the measurement setup and includes noise added by the digital sampling or low pass filters that are inevitably added by the measurement setup. The assumed power model such as Hamming distance or Hamming weight is only an approximation of the real power consumption of the device. For example, in the Hamming distance power models it is assumed that a switching of 1 to 0 has the same power consumption as the switching from 0 to 1. In practice, a switching from 1 to 0 might have a slightly different power consumption than the switching from 0 to 1. This mismatch between the real power consumption and the used power model is called model matching noise. Algorithmic noise describes the power consumption caused by parts of the chip that are not part of the power model of the attack but run in parallel. In the case of the controlled PUF design, algorithmic noise could for example be the power consumption of the challenge generator if it runs in parallel to the PUF, state machines or unrelated parts of the chip such as communication logic.
It is usually assumed that each of these noise sources are independent and there is an additive effect between all the noise sources so that their overall effect can be approximated by a Gaussian distribution [74]. Physical noise and measurement noise can be reduced by averaging over several measurements. Algorithmic noise and model matching noise on the other hand can often not be reduced using averaging since the noise is usually constant for a given input. Only if the algorithmic noise is independent from the input to the target IP core, e.g. a global counter, can averaging be applied to reduce the signal-to-noise ratio.

Predicting the added noise is very difficult since it depends on so many factors. Therefore different noise levels are used in the experiments to show that even in the presence of substantial noise a side-channel attack is possible. The power traces in the remainder of this work are recorded as follows: At first the power traces with the (idealized) used power model are computed. In a 1-bit Hamming distance power model this means that if the current response is different from the previous response a 1 is assigned as the power value, otherwise a 0. To this noise-free power model Gaussian noise, denoted as $\mathcal{N}(\mu, \sigma^2)$, with a mean value of $\mu$ and standard deviation of $\sigma$ is added to simulate the various noise sources. Note that the mean value $\mu$ does not have any influence on the correlation coefficient and therefore in the following $\mathcal{N}(0, \sigma^2)$ is used. The metric $\mathcal{N}(0, \sigma^2)$ might not be very intuitive to the reader and therefore a second metric is used as well. The power consumption during the rising edge of the clock is in many designs roughly proportional to the number of switching registers. To get a rough idea of how much noise $\mathcal{N}(0, \sigma^2)$ is we also represent the noise as the amount of switching registers that would add algorithmic noise equivalent to $\mathcal{N}(0, \sigma^2)$. The amount of noise added by $n$ independently and randomly (with a probability of 50%) switching registers with an idealized power model is approximately Gaussian with a mean of $\mu = n \times 1/2$ and a standard deviation of $\sigma = \sqrt{(n \times 1/4)}$.
Table 6.1. Example correlation coefficients (cc) and corresponding noise levels, depicted as $\mathcal{N}(\mu_N, \sigma^2_N)$ and the number of corresponding noise registers, of CPA attacks on different architectures: a PIC16F886 microcontroller that is used in an RFID access control system [62], the Yubikey One-Time Password Token that uses AES [62], the DS2432 and DS28E01 SHA-1 HMAC protected EEPROM from Maxim Integrated [62], Virtex-4 and Virtex-5 bitstream encryption based on AES-256 [57], an AES-128 implementation on a 22nm Kintex-7 FPGA [37], and an EM attack on the contactless Mifare Desfire smartcard (potentially with some side-channel countermeasures) [8]. Please note that the noise levels are only approximations based on the CPA values provided by the cited papers and are only meant to give the reader a rough idea of the expected noise levels in a side-channel attack.

However, this notion is only used to provide a more intuitive way of expressing the added noise. The added Gaussian noise represents all of the different noise sources since, as mentioned, these noise sources are assumed to follow a Gaussian distribution. How high the noise will be for an actual device depends on too many factors to be able to give a concrete answer. Therefore we simulate different noise levels that we assume to be reasonable approximations of what can be expected in a real measurement. But to give the reader an idea of the typical amount of noise in a side-channel attack, Table 6.1 gives an overview of some successful CPA attacks and their corresponding correlation coefficient and the corresponding amount of noise. How to calculate the Gaussian noise for a given correlation coefficient was discussed in Section 2.1.2.
6.2.2 Combining CPA with Machine Learning

The previous Section showed that we can distinguish PUF models that have a high model accuracy from PUF models with a lower model accuracy using the correlation coefficient as a metric. This indicates that power measurements can be useful in attacking a PUF. However, due to noise it is not possible to directly read out specific response bits from the power measurements. Therefore, machine learning techniques such as SVM that require challenge and response pairs seem do not seem to be well suited. But there exists a machine learning technique that works very well with correlation power analysis: Evolution Strategies (ES). The idea of ES is to randomly generate PUF models with different delay values and then test which of these models perform best, i.e. which are the fittest. The fittest models are then kept as parents for the next generation, while the other models are discarded. In the next generation, children are derived from these parents by randomly modifying the delay values of the parent models. In the next step the fitness of these children is evaluated and new parents for the next generation are chosen from these children. The idea behind this approach is that by always keeping only the best PUF models, the PUF models gradually become more accurate. This process is repeated and eventually the PUF models model the PUF with a very high accuracy.

Using ES to attack PUFs is not new and ES has already been successfully applied to attack feed-forward Arbiter PUFs [71]. The advantage of ES is that it is not based on solving any equations. For ES to work it is only necessary to have a way to distinguish which PUF models from a given set of PUF models are the fittest. Typically, this is done by comparing the modeled response bits with the measured response bits. The models with the highest match rate (accuracy) are the fittest. But any other fitness test that can distinguish good PUF models from bad PUF models with a high probability can be used. As discussed, it is possible to use power measurements and the correlation coefficient to distinguish PUF models that have a
high model accuracy from models with a small model accuracy. Therefore, correlation
power analysis and ES can easily be combined: ES is used to generate potential PUF
models and correlation power analysis is used to test the fitness of these models. ES
also has the advantage that its random nature makes it quite resistant to noise. If
a good PUF model is falsely discarded and instead a worse PUF model is chosen as
a parent for the next generation, this will slow down the convergence to the optimal
solution. But as long as more good PUF models are chosen as parents than bad
models, the ES still works and can converge to a near optimal solution. This property
is very helpful for noisy environments such as the discussed power side-channel.

There are many different variants of ES that mainly differ in how the child in-
stances are derived from the parent instances. We tested several different methods
and parameters. The optimal strategy depends on many aspects such as number
of stages of the PUF, noise, available traces and the computation environment. I
tested the \((\mu/\gamma)\)-S approach without recombination with and without self-adoption.
This approach has previously been successfully applied to feed-forward Arbiter PUFs
in [71]. However, in my experiments the CMA-ES outperformed the \((\mu/\gamma)\)-ES and
performed very well among all ranges of noise level. CMA-ES uses a weighted re-
combination approach with self-adoption. Details of this machine learning attack can
be found e.g. in [35]. The same parameters as proposed in [35] were used with the
exception that we increased the child-population since we are dealing with a very
noisy environment.

6.2.3 Results

As mentioned in Section 6.2.1 the design can be attacked using two different
power models: A 1-bit Hamming weight (HW) power model in which each challenge
is independent from the previous challenge and a Hamming distance power model
(HD) that targets the shift registers where the responses are stored. Figure 6.5 shows
the result of a combined correlation CMA-ES (referred to as CCMA-ES from here after) using a 1-bit HW power model and a noise of $\mathcal{N}(0, 25)$ which is equivalent to roughly 100 randomly switching registers. In Figure 6.6 the same attack is shown with a 1-bit HD power model. The CCMA-ES is a non-deterministic method, hence, if run with the same inputs, it can yield different results. In the figures 100 independent runs with the same PUF instance and power simulations were executed. The best run achieved an accuracy of 93.5% for the 1-bit HW power model compared to an accuracy of 95.6% for the 1-bit HD power model. However, while all runs converged to a solution close to the maximum in the 1-bit HW power model, in the 1-bit HD power model several runs did not converge to a solution close to the maximum. Hence, while the HD power model achieves higher accuracies if a run converges, there is a much higher chance that a run does not converge for the HD power model compared to the HW power model.

Figure 6.5. Result of a CMA-ES with a 1-bit HW power model, 150k challenges and a noise of $\mathcal{N}(0, 25)$ which is equivalent to 100 switching registers.

The reason for this is that a single miss-predicted response bit actually influences two bits in the HD power model (the current and next value). This leads to a relation between the prediction accuracy and correlation coefficient that unlike the Hamming weight power model is not linear. Figure 6.8 shows the relationship of accuracy
Figure 6.6. Result of a CMA-ES with a 1-bit HD power model, 150k challenges and a noise of $\mathcal{N}(0, 25)$ which is equivalent to 100 switching registers.

Figure 6.7. Result of a CMA-ES with an 80-bit HD power model, 150k challenges and a noise of $\mathcal{N}(0, 25)$ which is equivalent to 100 switching registers.
versus correlation coefficient for the HW as well as HD power model. It is still true for the HD power model that a higher accuracy yields a higher correlation coefficient. However, this correlation coefficient increases slower for low accuracies compared to higher accuracies. What this means in practice is that the HD power model does not perform as good as the HW power model while the prediction accuracy is fairly low. But for higher accuracies the HD model actually outperforms the HW model since for higher accuracies the correlation coefficient increases faster in the HD model compared to the HW model. This trend is independent of how many bits are used for the HW power model, but an 80-bit HD model shows a higher variance from the ideal curve than a 1-bit HD model. This explains why the HD model achieves higher accuracies while simultaneously having a lower rate of runs that converge.

Figure 6.8. Relation between the correlation coefficient and the prediction accuracy for the Hamming weight power model as well as the Hamming distance power model. For this simulation 1 million random response bits were used and no noise was added.

If we assume that the result is stored in an 80-bit shift register then the power consumption during the storing of a response bit follows an 80-bit HD power model, not a 1-bit HD model. Using an 80-bit power model greatly increases the signal to
noise ratio. Figure 6.7 shows the same attack as before with the 80-bit HD power model. In this case accuracies of up to 99.9% are achieved. Therefore this power model is recommended in practice, assuming the design allows it. It is also important to note that in the controlled PUF a single execution of the protocol requires 80 responses, hence, a single measurement actually contains 80 challenges. Hence, for the example of 150k challenges this means that only around $150,000/80=1875$ measurements would be needed.

Figure 6.2.3 shows the result for the CMA-ES attack with different noise levels and different number of traces with this 80-bit HD power model. As one can see, using the 80-bit HD power model makes the attack very robust to noise. Even with a noise level equivalent to 5000 randomly switching registers an accuracy of 96.5% is achieved using only 120k challenges which can be collected with as few as 1500 measurements. Model accuracy is only one metric to determine the success of the attack. Another valid metric is the number of times the correct 80-bit string is predicted. The output of the controlled PUF is the hash value of 80 response bits and therefore an attacker can verify if he has predicted the correct 80-bit string. To correctly predict these strings is the ultimate goal of the attacker since this will enable him to impersonate the PUF device.

Furthermore, if the attacker manages to predict the PUF with a high enough accuracy to find a single 80-bit match, the attacker can perform a second machine learning attack that uses this success metric to achieve accuracy exceeding 99%. The idea of this second machine learning attack is simple: Instead of still relying on the (noisy) power side-channel to evaluate the fitness of the PUF models, the attacker uses the number of string matches to determine the fitness. Otherwise the same CMA-ES is used as with the power model. Since this string match analysis is noise free, much larger accuracies can be achieved than with the power side-channel. Given enough challenges, accuracies beyond 99.99% are achieved.
Figure 6.9. Result of 100 runs of an 80-bit CCMA-ES attack with different levels of noise. The noise level is expressed by the number of randomly switching registers to achieve the same amount of Gaussian noise. On the left the maximum achieved accuracy with 100 runs is shown while on the right the number of runs that achieved an accuracy high enough to find at least one string match is shown.

The most effective attack is therefore a two step approach: In the first stage a combined machine learning and power side-channel attack is performed to model the PUF with a large enough accuracy to predict some of the 80-bit strings. In the second step, a machine learning attack using the number of string matches as a fitness function is used to achieve prediction accuracies beyond 99%. Figure 6.10 shows the required number of 80-bits strings for different PUF accuracies to find a match with a probability of at least 50%. This figure can be used to roughly determine the needed model accuracy that needs to be achieved using the power side-channel attack so that in a second step a CMA-ES based on string matches can be performed. For example, with an accuracy of 90% only about 10k traces are needed to find a string match. To reliably find a string match with 1 million traces an accuracy of 85% or better is required.

I tested this two-step approach by adding noise to the power measurement equivalent to a million random switching registers and the first step based on the power consumption achieved a model accuracy of 88%, which was enough to launch a sec-
Figure 6.10. The number of needed strings so that the probability of a match is at least 50%.

Second machine learning attack based on string matches with this output to successfully model the PUF with an accuracy of 99.99% using 10 million challenges (for which only 128k measurements are needed). Table 6.2 gives an overview of the required number of traces for different noise levels so that at least one string match is found. In each case it was then possible to achieve accuracies beyond 99% with the second machine learning attack. Table 6.3 shows the same analysis for the 1-bit HW power model. These noise levels are very large while the required number of traces is comparably small for a side-channel attack. Hence, it is reasonable to say that power side-channel attacks on controlled PUFs are successful even in the presence of considerable noise.

6.3 Fault Attack on Arbiter PUFs

So far we have seen that Arbiter PUFs are vulnerable to passive side-channel attacks. In this section we will take a closer look at their resistance against active attacks. It was often assumed that the fact that the PUF changes its behavior if it is being tampered with increases the security of PUFs against implementation attacks.
Table 6.2. Required number of challenges for different noise levels to achieve an accuracy large enough to find a string match with an 80-Bit HD power model. With such a string match a second machine learning algorithm achieved accuracies beyond 99%.

<table>
<thead>
<tr>
<th>Noise Registers</th>
<th>Challenges</th>
<th>Traces</th>
<th>Accuracy</th>
<th>String match</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>30,000</td>
<td>375</td>
<td>97</td>
<td>Yes</td>
</tr>
<tr>
<td>100</td>
<td>30,000</td>
<td>375</td>
<td>97</td>
<td>Yes</td>
</tr>
<tr>
<td>1,000</td>
<td>50,000</td>
<td>625</td>
<td>95</td>
<td>Yes</td>
</tr>
<tr>
<td>10,000</td>
<td>150,000</td>
<td>1,875</td>
<td>93</td>
<td>Yes</td>
</tr>
<tr>
<td>100,000</td>
<td>1,000,000</td>
<td>12,500</td>
<td>93.6</td>
<td>Yes</td>
</tr>
<tr>
<td>1,000,000</td>
<td>10,000,000</td>
<td>125,000</td>
<td>88.2</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 6.3. Required number of challenges for different noise levels to achieve an accuracy large enough to find a string match with a 1-Bit HW power model. With such a string match a second machine learning algorithm achieved accuracies beyond 99%.

<table>
<thead>
<tr>
<th>Noise Registers</th>
<th>Challenges</th>
<th>Traces</th>
<th>Accuracy</th>
<th>String match</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>80,000</td>
<td>1,000</td>
<td>89</td>
<td>Yes</td>
</tr>
<tr>
<td>500</td>
<td>450,000</td>
<td>5,625</td>
<td>90</td>
<td>Yes</td>
</tr>
<tr>
<td>1,000</td>
<td>750,000</td>
<td>9,375</td>
<td>89</td>
<td>Yes</td>
</tr>
<tr>
<td>5,000</td>
<td>4,000,000</td>
<td>50,000</td>
<td>87</td>
<td>Yes</td>
</tr>
<tr>
<td>10,000</td>
<td>7,500,000</td>
<td>93,750</td>
<td>88</td>
<td>Yes</td>
</tr>
</tbody>
</table>
However, in this Section we will see that the unreliability information can instead be used to successfully model a controlled PUF.

6.3.1 Impact of Noise on Arbiter PUFs

Arbiter PUFs have a problem with unreliability in practice, which means that the same challenge does not always generate the same response. There are two sources that can cause an Arbiter PUF to change its response bit: thermal noise and changes in environmental conditions. Thermal noise adds approximately Gaussian noise to the delay value of each individual execution of the PUF. If the delay difference between the top and bottom signal is small for a given challenge, this noise can switch the response bit by changing a previously positive delay difference into a negative delay difference and vice versa. The closer the delay difference is to zero, the more likely it is that the response bit changes.

The second reason why a response bit might flip is due to changes in the environment. For example, it is well known that changing the supply voltage or operation temperature changes the propagation delay and rise and fall time of a CMOS transistor. The magnitude of this change depends directly on the transistor sizing as well as the process variations. Hence, the Arbiter PUF behaves differently at different supply voltages. For each challenge the delay difference will either increase or decrease when the supply voltage is reduced. The amount of the increase or decrease depends on the PUF instance as well as the challenge.

This has some interesting implications. First of all, the attack by Delvaux et.al. [25] does not work with environmental noise in the same way as it does with thermal noise. The added delay is not Gaussian but a function of the supply voltage (or other environmental conditions) whose slope depends on the PUF instance as well as the challenge. Since the slope of the function is different for each challenge two challenges that have the same delay difference might behave differently when the supply
voltage is altered. One challenge might flip while the other does not. Nevertheless, if the delay difference is close to 0 it is much more likely that the response bit flips than if the absolute delay difference is large. In Figure 6.11 the delay differences for an 128-bit Arbiter PUF is shown. The delay difference is approximately Gaussian and ranges between -150ps and +150ps. When the supply voltage was altered from the nominal voltage of 1.1V to 1V and 1.2V roughly 6% of the response bits flipped. In Figure 6.11 the delay difference of the challenges that flipped are depicted in black. Only traces that were close to zero (between -12ps and +13 ps) flipped. The distribution of the delay difference of the flipped traces is again approximately Gaussian and the closer to zero the more likely that the response flipped. However, while a trace whose absolute delay difference was larger than 13ps never flipped, a lot of traces whose delay difference were less than 13ps did not flip. Hence, it is not possible to directly read out the delay difference from the information when a challenge flips. However, we still get enough information to reliably model the PUF as we will see in the following.

![Figure 6.11](image)

**Figure 6.11.** The delay difference in pico seconds between the top and bottom signal after the last stage for 49k traces. Colored in blue are the delay differences of all traces and in black are the delay differences for the traces whose output flipped when the supply voltage was changed from 1.1V to 1V and 1.2V.
For the controlled PUF design from Section 6.1.3 to be useful an Arbiter PUF with a high reliability is needed. Ideally, the PUF should be resistant to thermal noise and changes in the environmental conditions. However, this is very difficult to achieve for all possible environmental conditions. Therefore, it is much more likely that the PUF will be designed to be reliable under normal operation conditions as defined in the specifications. For example, techniques such as the one described in [23] can help to counteract changes in temperature or supply voltage. Usually, these techniques are optimized for a small range of operating conditions and will not work outside the specification of the device. Furthermore, it is likely that the controlled PUF will depend on a stable voltage source or that the protocol allows to collect challenges and responses for different working conditions. Hence, even if a controlled PUF might be very reliable under normal working conditions, it is reasonable to assume that an attacker with access to the device can alter the environmental conditions to make the PUF unreliable. In particular, it is likely that an attacker with physical access to the PUF is able to apply a supply voltage that is outside of the specifications.

In the following, we will again assume that a controlled PUF design as described in Section 6.1.3 is used. We furthermore assume that the attacker can alter the voltage supply in a way that the chip is still functioning but some PUF responses will flip. The advantage of changing the supply voltage compared to e.g. changing the temperature is that it can be done easily in a fast and automatic fashion. It is possible to build a set-up that will alter the supply voltage of the chip for only a few clock cycles. This has the advantage that only one or a few challenges of the challenges in the controlled PUF design will be subject to the altered supply voltage. Other environmental changes, e.g. generating a strong magnetic field might have the same or similar effects. What kind of faults are feasible and effective besides voltage manipulations is an interesting future research direction. Basically any change to the PUF that can be applied to a single challenge (or few challenges) and that changes
the delay difference so that only a subset of response bits flip can be used for our fault attack.

We performed Monte Carlo simulations using HSpice for an 128-bit Arbiter PUF in 45nm technology with a nominal voltage of 1.1V and with 1V and 1.2V and compared the response bits to determine which responses flip when the voltage is altered. It turns out that when reducing the voltage from 1.1V to 1V only 1.9% of the responses flipped while increasing the voltage from 1.1V to 1.2V resulted in 4.5% of flipped bits. One reason why increasing the voltage had larger effects on the responses is that the standard cells we used were optimized for usage of 1.1V and 1V but not for 1.2V\(^3\). In total 6% of the challenges flipped and only 0.4% of the challenges flipped for both a voltage reduction as well as voltage increase. In the following, data from these simulations are used to test our fault attack.

6.3.2 Combined Machine Learning Fault Attack

The same CMA-ES machine learning method as in the combined power side-channel and machine learning attack is used for the combined machine learning fault attack. Recall that it is necessary to be able to distinguish PUF models that have a high model accuracy from PUF models with a low model accuracy for an ES machine learning attack to work. In Section 6.2.2 this was done using the power side-channel. Instead, in this attack the reliability information of the PUF under supply voltage manipulations is used to judge the PUF models. The basic idea is to take measurements of the same challenges with different supply voltages and check for which challenges the response changed. If the response bit for challenge \(i\) flipped we assign \(F_i = 1\) otherwise \(F_i = 0\). From Figure 6.11 we know that if a response bit flipped it is very likely that the corresponding delay difference was close to zero i.e. \(|\Delta D| < \tau\).

\(^3\)Note that this can also be due to the fact that the HSpice simulations are more accurate the closer you simulate to the nominal operation conditions.
To test a PUF model we compute the delay difference $\Delta \hat{D}_i$ for each challenge $i$ and assign:

$$\hat{F}_i = \begin{cases} 
1 & \text{if } |\Delta \hat{D}_i| < \hat{\tau} \\
0 & \text{if } |\Delta \hat{D}_i| > \hat{\tau}
\end{cases}$$

One way to check the fitness of the PUF model is to compare the hypothesis vector $\hat{F}$ with the measured vector $F$ by counting how often the two vectors are equal, i.e., $\sum |F_i - \hat{F}_i|$. Intuitively, the correct PUF model should have the smallest difference between the two vectors. However, using the number of mismatches is not very good to measure the fitness of the PUF instances. The number of flipped bits is relatively small and as seen in Figure 6.11, not all bits flip if $|\Delta D| < \tau$. Hence, even for the correct PUF model there is a mismatch between $F$ and $\hat{F}$, i.e. $\sum |F_i - \hat{F}_i| > 0$. A PUF model that results in a vector $\hat{F}_i = 0$ for all $i$ might therefore have a smaller mismatch between $\hat{F}$ and $F$ than a PUF model with a high model accuracy. Therefore, using the number of mismatches as the fitness test does not work in practice. Using the correlation coefficient between $F$ and $\hat{F}$ on the other hand works very well. Hence, the correlation coefficient is used again to determine the fitness of the PUF models.

Compared to the CMA-ES based on power side-channels, the fault based CMA-ES needs to model the parameter $\tau$ in addition to the $n+1$ delay values. But otherwise there is not much difference between the two attacks.

Figure 6.12 shows the result of the fault CMA-ES with 8k traces and +/- 0.1V supply voltage variation. The attack can model the PUF reliably with an accuracy of 97.7%. From 100 runs 56 runs achieved an accuracy of at least 97% with a computation time of 70 minutes for all 100 runs. These results represent an ideal case without any noise. However, when performing this attack in practice there might be measurement errors in the fault trace $F$. In the controlled PUF case we assume that we can alter the operating conditions for only a single sub-challenge while keeping the operating conditions constant for the other 79 sub-challenges. Since the 80 response
Figure 6.12. The result of 100 runs of the CMA-ES Fault attack with 8k traces and $+/-0.1V$ supply voltage variation without additional noise. On the left the accuracy of the resulting PUF models are shown. On the right side the number of 80-bit strings that were correctly predicted by the PUF models are shown.

Figure 6.13. The result of 100 runs of the CMA-ES Fault attack with 45k traces and $+/-0.1V$ supply voltage variation with 20% faulty responses.

bits are fed through a hash function we can see if at least one response bit flipped by checking if the hash value changed. But since a hash function is used, it is not possible to determine which response bit has changed or how many. It can therefore happen that a response bit flip is detected although it actually corresponds to a flip in one of the 79 sub-challenges that are not targeted. We call this case a “false positive” since $F_i$ is set to one although no flip actually happened for the targeted challenge.

Besides a “false positive” there could also be a “false negative” in which a response bit that is supposed to flip does not flip, e.g. because the fault setup did not work
correctly. It is likely that the closer $|\Delta D|$ is to zero the more unlikely it is that a false negative happens for this challenge. False negatives as well as false positives can be reduced by repeating the measurement several times. It could also happen that the targeted challenge flips due to noise instead of the voltage alteration. However, if a challenge flips due to noise it is likely that the delay difference is close to zero as well, i.e. $|\Delta D| < \tau_n$. Hence, whether the response flips due to noise or voltage alteration does not matter as long as $\tau_n \leq \tau$.

We tested how the attack works in the presence of these noise sources. We randomly changed 20% of the bits of $F$ which represents a false positive rate of 20% and a false negative rate of 20%\(^4\). Figure 6.13 shows the result of a CMA-ES attack with 45k challenges with 20% overall noise. The attack still works and achieves the same accuracy as the noiseless case. However, the number of challenges needed to attack the PUF increased from 8k challenges to 45k challenges and more runs are unsuccessful than in the noise-free case. Nevertheless, the number of challenges is still fairly low considering the rather large amount of noise added. Figure 6.14 shows the success of the attack with different noise levels and different numbers of challenges.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{result.png}
\caption{Result of a Fault CMA-ES attack with 200 independent runs for different levels of overall-noise and for different numbers of challenges. On the left the highest achieved model accuracy is shown while the right figure shows the number of runs that had at least one string match.}
\end{figure}

\footnote{Since we change the bits independent of $|\Delta D|$ this can be seen as a worst case scenario.}
As can be seen, the attack still works in the presence of considerable noise. Hence, fault attacks on controlled PUFs based on Arbiter PUFs are feasible. The presumed strength, that tampering with the PUF results in different responses, turns out to actually be a security vulnerability. One can therefore no longer state that Arbiter PUFs are more resistant to implementation and probing attacks than traditional cryptography.

6.4 Implications of Side-Channel Attacks on Arbiter PUFs

The results presented in this chapter show that — unlike often believed — Arbiter PUFs are not considerably more resistant against side-channel attacks than traditional cryptographic algorithms. So where does this leave us? We already knew that Arbiter PUFs have some serious problems with machine learning attacks and reliability. But due to the nice property that PUFs work without the need to program a key, their lightweight nature in terms of area and power, and their presumable resistance to implementation attacks, Arbiter PUF have gained a lot of research attention despite these known problems. However, now that we know that this assumed strength against implementation attacks does not hold, are Arbiter PUFs still a promising solution?

Without adding considerable non-linearity to the Arbiter PUFs, the problem of machine learning attacks remain. But when adding such non-linearity e.g. by using a controlled PUF or by adding a high amount of XOR PUFs, the Arbiter PUF looses much of its lightweight nature. And there already exist very lightweight cryptographic algorithms as alternatives such as lightweight block ciphers or stream ciphers. This raises strong doubts whether Arbiter PUFs are significantly more lightweight than traditional cryptographic solutions.

\footnote{It is also possible to attack the PUF by probing a digital signal in the post-processing of the controlled PUF design.}
Therefore, the only remaining advantage of Arbiter PUF compared to traditional cryptography is its property that the “secret” does not need to be programed nor that secure non-volatile memory is needed to store a key. This is indeed a nice property e.g. in a product authentication setting. However, for the application of generating keys, weak PUFs can be used as well. Hence, the competition for Arbiter PUF in this case is not traditional cryptography but weak PUFs such as Ring-Oscillator PUFs and memory based PUFs. That weak PUFs are a promising security primitive is known, with commercial solutions already available.

So what is the advantage of using an Arbiter PUF compared to using a weak PUF combined with a traditional cryptographic algorithm such as a hash function to establish a challenge and response protocol? It seems that using a weak PUF combined with traditional cryptography is a much more solid solution with a much higher understanding of the security implications. Of course, Arbiter PUFs can also be used as weak PUFs to generate cryptographic keys, but then they should also be treated as weak PUFs and not a strong PUFs.

When Arbiter PUFs were first proposed, it boosted a whole new research area: that of strong PUFs. The idea was very promising, but we should admit that Arbiter PUFs have too many problems to be a secure and reliable solution for challenge-and-response protocols. It is about time that the research community admits that Arbiter PUFs are actually weak PUFs — not strong PUFs — and cannot be used for challenge-and-response protocols. Unfortunately, no alternative to delay based PUFs as an electrical strong PUF has emerged so far. Instead of focusing on improving a failed method, the research should focus on finding alternative and novel ideas to build strong PUFs. Otherwise, it is much more promising to research how weak PUFs can be combined with traditional cryptography to solve the problems that were hoped to be solved with strong PUFs. Such a solution is much more promising than trying to fix something that is broken by design.
CHAPTER 7
CONCLUSION

The heavy use of outsourcing and globalization in the embedded system design process brings many security challenges. This thesis shows that side-channel analysis can be a tool for a malicious attacker to build stealthy hardware Trojans but also a tool that can be used to protect your intellectual property in embedded systems. IP theft is a serious concern for companies working in the embedded systems market. The business model of many companies in the embedded systems market is to sell IP-cores or software libraries for embedded systems. However, often these companies have to provided access to their IP-cores and libraries to potential customers since the costumers often want to test these designs. But companies have no guarantee that these costumers handle the IP responsible and that they do not use them unless they paid for it. Detecting IP theft and proofing this misuse is very difficult in embedded systems. The side-channel based hardware and software watermarks presented in this thesis address this issue. The major advantage of these side-channel based watermarks is that it is possible to verify the presence of the watermark without access to the design details of the embedded system. All that is required to detect the watermark is access to the chip under test to perform power or EM side-channel measurements. This feature makes side-channels a perfect candidate to detect IP theft in embedded systems, since a verifier usually does not have access to the design details of the device under test.

But IP theft is not the only problem that the outsourcing and globalization of the embedded system design process has brought. Hardware Trojans are an increasing
concern. One fear is that foreign governments might put pressure on companies to insert back doors into the embedded systems. Most companies in the embedded system market are “fab-less”, i.e., they do not have their own factories but outsource the manufacturing process to a different company. Often these factories are located in different countries. As the Snowden revelations about the practices of the NSA show, some governments put pressure on their IT industry to build in backdoors for their secret agencies. Although no reports are known in which a semiconductor factory inserted a hardware Trojan, it seems that this is a realistic threat.

Therefore, reliable methods are needed to detect hardware Trojans inserted at the manufacturing level. But to build efficient detection mechanisms, one first needs to understand how hardware Trojans can be build. In this thesis an extremely stealthy method of how to build hardware Trojans is introduced. The dopant Trojans are realized by only modifying the design below the transistor level. No additional transistors or metal wires are inserted. Two case studies proof that this technique can be used to build meaningful hardware Trojans that will likely pass currently proposed Trojan detection mechanisms. These new Trojans give important insight in how stealthy hardware Trojans can be in practice and highlight that better Trojan detection mechanisms are needed.

Besides IP-theft and hardware Trojans, counterfeits are another major concern in the embedded system market. These counterfeits not only steal revenue from the manufacturer of the IC but also cause serious problems for the company who unknowingly uses them in their products. The counterfeit ICs are often much less reliable than the original ICs and often have different specifications compromising the performance and reliability of the entire end-product.

The side-channel based hardware watermarks used to detect IP theft can be extended to also be used to detect counterfeit ICs by storing the watermark in programmable memory. Another previously proposed anti-counterfeit mechanism is the
use of PUFs. For the anti-counterfeiting application a *strong PUF* is needed, i.e., a PUF with a challenge space large enough that trying all challenges is computationally infeasible. But building strong PUFs is very difficult and the currently most promising approach, the delay based PUFs, suffer from unreliability issues and, most importantly, vulnerability towards machine learning attacks. Nevertheless, Arbiter PUFs have gained a lot of attention. PUFs are often assumed to be more resistant against side-channel attacks, making them a very interesting for embedded systems. However, this side-channel resistance was never proven. As it turns out, by combining side-channel analysis with machine learning, it is possible to perform power side-channel attacks as well as fault attacks on Arbiter PUFs. The results presented in this paper show that even Arbiter designs that are resistant against machine learning attacks, such as a controlled PUF, are vulnerable. However, if delay based PUFs are not more resistant against implementation attacks than traditional cryptography, they loose much of their presumed strength. It is therefore the question if delay based PUF really should be considered strong PUFs and if they really are a promising security solution.

In summary, side-channel analysis is an extremely versatile tool with many malicious, but also constructive uses in embedded systems.


