Advanced Methods in Side Channel Cryptanalysis

Kai Schramm

July 2006

Dissertation for the Degree of Doktor Ingenieur

Department for Electrical Engineering and Information Technology University of Bochum in Germany

Communication Security (COSY) Group
Supervisor: Prof. Dr.-Ing. Christof Paar
External Referee: Prof. David Naccache
"I know that I do not know anything, but I am searching for the truth."
Sokrates (468 B.C. - 399 B.C.)
Dedicated to my parents for their continuous love and support.
Abstract

Cryptographers have traditionally designed new ciphers under the fundamental assumption that an implementation is realized in a closed, reliable and tamper resistant environment which does not leak any secret information. However, in reality, ciphers are typically either implemented in software, e.g. as assembly programs executed by a microprocessor, as dedicated hardware, or as a combination of both. Unfortunately, physical implementations of ciphers often tend to leak unintentional and sensitive information about their internal states. Sensitive, i.e. key-related, information may thus leak through a side channel, such as the power consumption, electromagnetic radiation or execution time. These side channels can be passively observed by an adversary without the need to tamper or manipulate the actual implementation.

Side channel cryptanalysis attempts to exploit physical leakages of a target device in order to extract the secret keys used in the implemented cipher. The research topic side channel cryptanalysis clearly shows that an insecure or flawed implementation which ignores side channel attacks will open up severe security weaknesses even if the implemented algorithm is perfectly secure against pure mathematical attacks. If cryptographers and design engineers fail to cooperate and do not mutually check each other's work, a security system is very likely to display some inherent vulnerability. Hence, the interaction of differently trained people plays an essential role during the design process of security systems.

This thesis provides a detailed insight into new and refined side channel attacks, and corresponding countermeasures. First, it is shown that key-dependent, internal collisions may occur within block ciphers and that such collisions can be detected by side channel analysis. As a proof of concept, software implementations of the Data Encryption Standard (DES) and Advanced Encryption Standard (AES) are successfully attacked. Furthermore, it is shown that internal collisions can also be exploited in the block ciphers Serpent and Kasumi. The practicability of these attacks stresses that countermeasures are indispensable.

Motivated by this fact, the random masking of intermediate, secret data in ciphers is thoroughly analyzed as a countermeasure against side channel attacks. Using the AES as a case study, different variants of the masking countermeasure are compared and evaluated. While it may seem straightforward that the masking countermeasure prevents side channel attacks, several security vulnerabilities are discussed which may occur in both software and hardware implementations, if the countermeasure is not carefully implemented. For instance, it is shown that glitches in masked CMOS logic circuits, e.g. a masked AES S-box, foil the notion of randomized blinding and allow first order side channel attacks.
Moreover, a new attack is proposed which combines classic Differential Power Analysis (DPA) with multivariate signal classification. This attack evaluates measured side channel traces in order to generate so-called templates which describe the multivariate leakage characteristics of the target device. Unlike classic DPA the attack consists of two phases and presumes that the adversary possesses a free-programmable test device identical to the target device. If the two devices are protected by the masking countermeasure and if the Random Number Generator (RNG) of the test device is biased or has intentionally been destroyed, then it is shown that the target device can be broken, even if the RNG of the target device is fully functional. Thus the attack opens up a potential back door for smart card manufacturers, vendors and developers, because it gives them the possibility to break their own implementations, if they have access to a smart card, whose RNG is biased, defective or has been intentionally destroyed. As a proof of concept, the results of successful attacks against masked smart card implementations of the DES and AES are given.
Deutsche Kurzfassung


 Die Seitenkanalkryptanalyse versucht physikalische Informationsquellen einer kryptografischen Implementierung mit statistischen Verfahren auszuwerten um geheime Schlüsselinformationen zu extrahieren. Das Forschungsthema Seitenkanalkryptanalyse verdeutlicht dabei vor allem, dass eine unsichere und fehlerhafte Implementierung, welche Seitenkanalattacken nicht berücksichtigt, erhebliche Sicherheitsmängel aufweisen wird selbst wenn der implementierte kryptografische Algorithmus mathematisch nicht gebrochen werden kann. Daher ist eine enge Zusammenarbeit zwischen Kryptografen und Entwicklungssingenieuren, welche ein entsprechendes Verfahren implementieren, in der Regel unverzichtbar.


hingewiesen, das sogenannte Glitches, d.h. mehrfache Schaltvorgänge am Ausgang eines Logikgatters während eines Taktzyklus, zu einem erheblichen Sicherheitsrisiko in maskierten Hardwareimplementierungen führen können.

Acknowledgements

I would like to express my gratitude to various different people. Clearly, I have to begin with my supervisor and the head of the communication security (COSY) group and the Horst Görtz Institute for IT security (HGI) at the university of Bochum in Germany, Professor Christof Paar. Christof has the extraordinary talent to motivate and inspire even if oneself is ready to give up. Christof’s support has certainly helped me to keep researching in the area of side channel cryptanalysis even though in the beginning this was a very new topic in our group and there was not a lot of knowledge about it. Next, I would like to thank our team assistant (and the true boss ;-) Irmgard Kühn. Without Irmgard our group would certainly sink in a world of chaos, because most of us (myself included) seem to lack any organisational skills. Then, I would like to thank the three former Ph.D. graduates Thomas Wollinger, André Weimerskirch and Jorge Guajardo, who I looked up to when I first became a Ph.D. student. In particular, I have to thank Thomas, who supervised my Master’s thesis on internal collisions in the DES block cipher. I believe I learned quite a bit from Thomas’ calm way to approach new problems. As a result, we were able to publish the results of my Master’s thesis at FSE 2003. I also have to thank Jan Pelzl and Sandeep Kumar. The three of us began as Ph.D. students in October 2002 and we all graduated in June/July 2006. During the last three and a half years both Jan and Sandeep have become very good friends of mine. I think I will miss both Jan as a drinking buddy who is always willing to enjoy a Fiege beer and also Sandeep’s own sarcastic sense of humor. Of course, I have to thank all the people who joined our group afterwards, i.e. Andy Rupp, Kerstin Lemke, the mad professor Ahmad Sadeghi and the next generation of Ph.D. students who recently began in our group, i.e., Tim Güneysu, Axel Poschmann and our future side channel guru Thomas Eisenbarth.

Furthermore, I would like to thank Patrick Felke and Gregor Leander, who are Ph.D. graduates in mathematics and cryptography of Professor Hans Dobbertin at Bochum university. We had many good discussions on internal collisions in the AES block cipher and were able to publish our joint work at CHES 2004.

In the late summer of 2004 from August to October I had the great opportunity to spend the summer at the IBM T.J. Watson Research Center in Hawthorne, NY, USA. I would like to express my gratitude to my immediate supervisor and mentor Pankaj Rohatgi, Dakshi Agrawal and my manager J.R. Rao. During this time I had many enlightening discussions with my supervisor Pankaj. In particular, we investigated new applications of template attacks which are based on multivariate analysis of side channel signals. Pankaj had several excellent ideas and I was very fortunate to learn from him. Moreover, he often also discussed various other topics, for example the meaning of statistics and uniformity in cryptography. Finally, we were able to publish the harvest of our work at CHES 2005.
Approximately one year later in the late summer of 2005 from August to October I was again extremely fortunate to work in an external research institute. During this time I worked at the Hitachi Central Research Laboratory in Nishi-Kokubunji, Tokyo, Japan. In particular I would like to express my deep gratitude to my manager Toshio Okochi (Okochi-san) and my immediate colleagues Takashi Watanabe (Watanabe-san) and Takashi Endo (Endo-san) for letting me work with them. We had many interesting and good discussions and both Watanabe-san and Endo-san helped me very much to understand and explore the glitching activities which occur in masked hardware implementations of cryptographic algorithms.

Finally, I would like to thank Elisabeth Oswald and Stefan Mangard from the Institute of Applied Information Processing and Communications (IAIK) in Graz, Austria. I had very good and helpful discussions with Elisabeth and Stefan on various topics in side channel cryptanalysis. I was very fortunate to cooperate with both of them which resulted in the publication of a joint work together with Elisabeth at WISA 2005 about software implementations of the masked AES S-box based on inversions in the composite field. Furthermore, I published a joint work together with Stefan on the vulnerabilities of masked AES S-box hardware implementations due to different signal propagation delays and glitching activities at CHES 2006.
Contents

1. Introduction ................................. 1
   1.1. Motivation .................................. 1
   1.2. Summary of Research Contribution ............. 2

2. Overview of Side Channel Cryptanalysis ............. 5
   2.1. Simple Power Analysis ........................ 6
   2.2. Differential Power Analysis .................... 9
   2.3. Higher Order Differential Power Analysis ......... 12
   2.4. T-Test .................................. 13
   2.5. Void Hypothesis DPA .......................... 14
   2.6. Correlation Coefficient ......................... 14
   2.7. Template Attacks ............................. 17

3. Internal Collision Attacks ......................... 19
   3.1. Previous Work ................................ 19
   3.2. Internal Collisions in DES ...................... 22
      3.2.1. Collisions in a Single S-box ................. 22
      3.2.2. Collisions in Three Adjacent S-boxes .......... 23
      3.2.3. Optimization of the Collision Attack against DES .... 27
      3.2.4. Attacking a Software Implementation of DES ........ 31
      3.2.5. A Cryptographically Strengthened S-box Resistant to Collision Attacks ........ 35
   3.3. Internal Collisions in AES ..................... 38
      3.3.1. Collisions in the MixColumn Transformation .... 38
3.3.2. An Analysis of the Collision Function ......................... 39
3.3.3. Optimization of the AES Collision Attack .................... 43
3.3.4. Simulation and Practical Attack ............................... 46
3.4. Internal Collisions in Serpent ................................. 48
  3.4.1. The Serpent Algorithm .................................. 48
  3.4.2. Partial Collisions in the Linear Transformation .......... 50
3.5. Internal Collisions in Kasumi ................................. 55
  3.5.1. Brief Overview of KASUMI .............................. 55
  3.5.2. Collisions in Function FL ............................. 60
  3.5.3. Collisions in Function FO ............................. 60

4. Masking Strategies for AES .......................... 67
  4.1. Previous Work ............................................ 67
  4.2. AES AVR Smart Card Implementation .......................... 71
    4.2.1. Atmel ATM163 microcontroller .......................... 71
    4.2.2. Simple Operating System for Smartcard Education (SOSSE) .. 72
    4.2.3. Properties of the AES .................................. 73
    4.2.4. Reference Implementation of the AES ................... 74
  4.3. First Order Masking of AES ................................ 74
    4.3.1. Composite Field Based Inversion ....................... 74
    4.3.2. Implementation of the Composite Field Masking Scheme .... 79
    4.3.3. Power Analysis of the new Scheme ..................... 80
  4.4. Higher Order Masking of AES ............................... 81
    4.4.1. HODPA: Theoretical Issues ............................ 82
    4.4.2. Multi-Bit SODPA of the AES S-box Input ................ 82
    4.4.3. Multi-Bit HODPA of the AES S-box Output ................ 85
    4.4.4. Single-Bit HODPA of the AES S-box Output .............. 90
    4.4.5. Secure HODPA AES Masking scheme ...................... 93
    4.4.6. S-box Recomputation Algorithm ........................ 96
    4.4.7. Mask Propagation and the MixColumn Transformation .... 98
    4.4.8. HODPA-Resistant AES Implementations ................ 99
5. Template-Enhanced DPA .......................... 101
   5.1. Previous Work .................................. 101
   5.2. Single-Bit Template Classification .................. 103
   5.3. Breaking the Masking Countermeasure: Template-Enhanced DPA ... 105
      5.3.1. Overview ................................... 105
      5.3.2. Profiling Phase ............................ 107
      5.3.3. Hypothesis Testing Phase .................... 109
      5.3.4. Experimental Results ....................... 110
      5.3.5. Sensitivity of classification rate on RNG bias ...... 112

6. Vulnerabilities of Masked AES Hardware Implementations ............... 119
   6.1. Previous Work .................................... 119
   6.2. Attacks on Masked AES Hardware Implementations ......... 121
      6.2.1. Zero-Offset DPA ............................. 122
      6.2.2. Toggle-Count DPA ............................ 125
      6.2.3. Zero-Input DPA ............................. 127
   6.3. Pinpointing the side channel Leakage of Masked S-boxes ........ 128
      6.3.1. Masked AND Gate ............................ 130
      6.3.2. Masked Multipliers for $GF(2^2)$ and $GF(2^4)$ ...... 131
      6.3.3. Masked AES S-boxes ........................ 132
   6.4. Countermeasures ................................ 134

7. Conclusions and Future Work ................................ 137

A. Bibliography ........................................ 141
List of Tables

3.1. Maximum number of S-box Inputs which result in a collision at the output of a single S-box for input differentials $\delta_j, \delta_{j+1}, \delta_{j+2}$. .......................... 25
3.2. Maximum Probabilities of Collisions at the Output of S-box Triplets. .......................... 25
3.3. Internal collision attacks against DES. ......................................................... 28
3.4. Best results of internal collision attacks against all eight possible S-box triplets. Each attack used $v$ 18-bit input differentials $\Delta_i$ and on average revealed $K$ key candidates after $C$ encryptions. The results were averaged over $10^4$ random keys. ......................................................... 31
3.5. Maximum biases $S2_{max}$ of the original eight DES S-boxes. .......................... 36
3.6. Strengthened DES S-box resistant against internal collision attacks. ....... 37
3.7. Probability of a collision after $n$ encryptions .......................... 43
3.8. Average no. of key candidates after one or more collisions have occurred. .. 47
3.9. The eight non-linear, bijective 4-bit S-boxes used in Serpent. ....... 50
3.10. The linear transformation step computes every output bit by an X-or addition of several input bits. ......................................................... 51
3.11. Number of output bits of the linear transformation in Serpent, which change, if a particular input bit changes. The indices of the 128 input bits are given in hexadecimal notation. ......................................................... 52
3.12. Distribution of 4-bit output differentials $\delta$ for a given 4-bit input differential $\epsilon$ with regard to S-box $S_0$ of Serpent. ......................................................... 53
3.13. All possible combinations of two input bits which will only change one 4-bit output. ......................................................... 55
3.14. All possible combinations of three input bits which will only change one 4-bit output. ......................................................... 56
3.15. All possible combinations of four input bits which will only change one 4-bit output. ......................................................... 56
3.16. All possible combinations of five input bits which will only change one 4-bit output. ................................................. 57
3.17. All possible combinations of six input bits which will only change one 4-bit output. ................................................. 57
3.18. All possible combinations of seven input bits which will only change one 4-bit output. ................................................. 58
3.19. KASUMI S-box $S7$: For every differential pair $(\Phi_i, \Phi_o)$ with $\Phi_i = \{1\}$ there exist two solutions $(y, y \oplus \Phi_i)$ which fulfill the equation $\Phi_o = S7(y) \oplus S7(y \oplus \Phi_i)$ with $y, \Phi_i, \Phi_o \in GF(2^7)$. ................................................. 62
3.20. KASUMI S-box $S9$: For every differential pair $(\Delta_i, \Delta_o)$ there exist two solutions $(x, x \oplus \Delta_i)$ which fulfill the equation $\Delta_o = S9(x) \oplus S9(x \oplus \Delta_i)$ with $x, \Delta_i \in GF(2^9)$ and $\Delta_o \in GF(2^9)$. ................................................. 63
3.21. KASUMI S-box $S9$: input differentials $\Delta_i$ and output differentials $\Delta_o = S9(x) \oplus S9(x \oplus \Delta_i)$, $x \in \{256, ..., 512\}$. ................................................. 64
3.22. KASUMI S-box $S9$: input differentials $\Delta_i$ and output differentials $\Delta_o = S9(x) \oplus S9(x \oplus \Delta_i)$, $x \in \{256, ..., 512\}$. ................................................. 65

4.1. Reference implementation of the AES for the Atmel AVR ATM163 smart card (without any masking countermeasures). ................................................. 74
4.2. Comparison of various AES software implementations with regard to code size and speed for a single encryption. ................................................. 80
4.3. Correlation coefficients of a successful multi-bit DPA for a given order $d$ predicting the AES S-box output. The leakage is presumed to be equal to the Hamming weight of intermediate variables (no additive noise). ................................................. 87
4.4. Number of measurements $N$ required to achieve an $|SNR| \geq 5$ in simulated multi-bit DPA attacks for a given order $d$ predicting the AES S-box output. (averaged over 100 simulated DPA attacks for each order $d$). The leakage is presumed to be equal to the Hamming weight of intermediate variables (no noise). ................................................. 88
4.5. Assessed number of measurements of HODPA attacks using the Fisher Z-transformation and the confidence interval $P(d > 0) = 0.9999$. The leakage is presumed to be equal to the Hamming weight of intermediate variables (no noise). ................................................. 90
4.6. Number of measurements $N$ required to achieve an $|SNR| \geq 5$ in simulated HODPA based on the HW-model (parameters: Offset = 10 mA, $\epsilon = 3.72$ mA, $\sigma = 1.9636$ mA) attacks (averaged over 100 simulated DPA attacks for each order $d$). ................................................. 90
4.7. Correlation coefficients of a successful single-bit HODPA for various orders \(d\) with parameters \(\epsilon = 3.1838\) mA and \(\sigma = 16.9143\) mA according to the general model. ..........................................
93

4.8. Number of measurements \(N\) required to achieve an \(|SNR|\) of \(\geq 5\) in simulated single-bit HODPA attacks with parameters \(\epsilon = 3.1838\) mA and \(\sigma = 16.9143\) mA (averaged over 100 simulated DPA attacks for each order \(d\)). ....................................................... 93

4.9. Assessed number of measurements of HODPA attacks using the Fisher Z-transformation and the confidence interval \(P(d > 0) = 0.9999\). The leakage is presumed to depend on the state of a single bit, only, with additive Gaussian noise. ........................................ 93

4.10. Overview of different S-box masking algorithms. ................................. 97

4.11. Details of various HODPA resistant AES AVR implementations. ............ 100

5.1. S-box output bit classification success rates \(\eta_{S,b_j}\) using templates built with 1400 samples and 50 significant points. .............................. 105

5.2. Number of measurements \(M\) required to achieve a constant SNR in a DPA differential trace for several RNG off-biases \(\nu\). ................................. 109

6.1. Results of various DPA attacks against a masked hardware implementation of AES. All attacks except for the zero-offset DPA were successful. . 128
List of Figures

1.1. Various classes of implementation attacks. .......................... 2

2.1. Side channel leakage of a cryptographic implementation. ........ 5
2.2. CMOS inverter driving a load capacitance. .......................... 7
2.3. Power analysis of a micro chip. ................................. 7
2.4. Hamming weight leakage of a microprocessor. .................... 8
2.5. Input and output of a DES S-box. .................................. 10
2.6. Difference of means vs. T-Test. ................................. 13

3.1. Collisions in a Feistel cipher. ...................................... 21
3.2. Collisions in a single S-box of DES. .............................. 23
3.3. Bit mask for a collision in a single S-box. ...................... 23
3.4. Bit mask for a collision in three adjacent S-boxes. ............ 24
3.5. Possible collision tests for $n = 3$ differentials .................. 28
3.6. Additional collisions in one of the three S-boxes while preserving the state of the two remaining S-boxes. .................. 30
3.7. Propagation path of an internal collision in the $f$-function of DES. . 32
3.8. Measurement setup for power analysis of an 8051 microcontroller. . 33
3.9. Power consumption of the microcontroller encrypting $x$ during the S-box look-up in round two. .......................... 34
3.10. Power consumption of the microcontroller encrypting $(x \oplus \Delta)$ during the S-box look-up in round two. The power trace differs from the trace shown above in the time frame between 0 and 1000 ns, which indicates that no collision occurred. .......................... 34
3.11. Deviation of power traces with $\delta = 1...255$ from the reference trace with $\delta = 0$. .......................... 48
3.12. Function $FL$ in KASUMI ........................................ 58
3.13. KASUMI Function $FO$ ........................................ 59
3.14. KASUMI Function $FI$ ........................................ 59

4.1. Wiring of the ATM163 and the external EEPROM with the smart card contact pads ........................................ 72
4.2. DPA of the AES with no active countermeasure .................. 81
4.3. DPA of the AES with our new masked s-box scheme ............ 81
4.4. Correlation plot of a simulated second-order DPA against the AES S-box output according to the HW-model with no noise ......... 87
4.5. Correlation plot of a simulated third-order DPA against the AES S-box output according to the HW-model with no noise .......... 87
4.6. Correlation plot of a simulated second-order DPA against the AES S-box output according to the HW-model (parameters: $Offset = 10$ mA, $\epsilon = 3.72$ mA, $\sigma = 1.9636$ mA). ........................................ 91
4.7. Correlation plot of a simulated third-order DPA against the AES S-box output according to the HW-model (parameters: $Offset = 10$ mA, $\epsilon = 3.72$ mA, $\sigma = 1.9636$ mA). ........................................ 91
4.8. Correlation plot of a simulated single-bit second-order DPA against the AES S-box output according to the general model (parameters: $\epsilon = 3.1838$ mA and $\sigma = 16.9143$ mA). ........................................ 94
4.9. Correlation plot of a simulated single-bit third-order DPA against the AES S-box output according to the HW-model (parameters: $\epsilon = 3.1838$ mA and $\sigma = 16.9143$ mA). ........................................ 94
4.10. Insecure AES masking scheme using the same $d - 1$ input and output masks to thwart DPA attacks of order $d$ .................. 95
4.11. Secure AES masking scheme which uses $d - 1$ different input and output masks for each S-box to thwart DPA attacks of order $d$ .................. 96

5.1. Improved DPA metric of S-box 1, bit 0 of a test device (smart card A) running DES ........................................ 104
5.2. Blinded S-box input and output bit with a random input mask bit $n$ and a random output mask bit $m$. ........................................ 107
5.3. Smartcard A: DPA of the masked s-box output bit using the test device and DPA of the mask bit using the target device. ................. 111
5.4. Smart card B: DPA of the masked s-box output bit using the test device and DPA of the mask bit using the target device. ................. 112
5.5. Smart card B: DPA of the masked s-box output bit using the test device and DPA of the mask bit using the target device (both with wrong hypothesis). ........................................ 113

5.6. Probability of correct classification versus RNG bias. .................. 114

6.1. Architecture of the masked AES hardware implementation. ............ 121

6.2. Average number of toggles in our masked S-box circuit. .................. 126

6.3. Correlation coefficients of the toggle-count DPA against the masked AES ASIC with 15,000 measurements. The correct key hypothesis (225) is clearly distinguishable from all false key hypotheses. ...................... 126

6.4. Correlation coefficients of a zero-input DPA against the masked AES ASIC with 30,000 measurements. The correct key hypothesis (225) is clearly distinguishable from the false correlation coefficients. .............. 127

6.5. Common architecture of a masked multiplier. .............................. 129

6.6. Secure architecture of a masked multiplier using delay chains. ......... 135
List of Figures
1. Introduction

1.1. Motivation

Cryptographers have traditionally designed new ciphers under the fundamental assumption that an implementation is realized in a closed, reliable and tamper resistant environment which does not leak any information about its internal states. However, in the real world ciphers are typically either implemented as software, e.g. in form of assembly programs executed by a microprocessor, as dedicated hardware, or as a combination of both. Unfortunately, physical implementations of ciphers often tend to leak unintentional information about their internal state. Sensitive, i.e. key-related, information may thus leak through the power consumption, electromagnetic radiation or timing behavior of a cipher implementation. These side channels can be passively observed without tampering with the actual implementation.

For example, it is a well known fact that the U.S. government has spent considerable resources in the classified TEMPEST program in order to prevent the leakage of sensitive information through electromagnetic radiation since the 1950’s [Joe00]. In 1996, Paul Kocher published an article about timing attacks against the public key algorithm RSA at the CRYPTO conference [Koc96]. Two years later in 1998, the world of cryptographers and related industrial companies which design and fabricate cryptographic tokens, such as smart cards, was even more shocked when Paul Kocher, Joshua Jaffe and Benjamin Jun published their pioneering technical report¹ "Differential Power Analysis" [KJJ99]. The authors demonstrated that they were able to extract the keys of several widely used smart cards by power consumption analysis. As a result, the exciting research area of side channel cryptanalysis was born. Since then, several articles have been published at conferences such as Cryptographic Hardware and Embedded Systems (CHES).

Side channel cryptanalysis makes one important point clear: a real-world security system always consists of various layers, quite similar to the classic layer representation of data package transportation in computer networks. If cryptographers, software developers and hardware engineers fail to cooperate and do not mutually check each other’s work, such a security system is very likely to display some inherent vulnerability. Hence, the

¹One year later this report was published at the CRYPTO 1999 conference.
interaction of differently trained and skilled people plays a profound role during the design process of security systems.

Side channel cryptanalysis belongs to the broader class of implementation attacks. As shown in Figure 1.1, there are many other invasive and non-invasive implementation attacks which actively manipulate or tamper with cryptographic implementations, e.g. by inserting faults at particular points in time on certain bus signals [ABF+02]. Also, semi-passive attacks which monitor and evaluate the internal bus signals of cryptographic implementations have been published [ISW03].

![Implementation Attacks Diagram](image)

**Figure 1.1:** Various classes of implementation attacks.

This thesis provides a detailed insight into new and refined side channel attacks. Passively monitoring the power consumption or electromagnetic radiation of a cryptographic device has some significant advantages over other implementation attacks: physical access to the target device can be restricted (e.g. when recording electromagnetic traces with an antenna from another room). Moreover, some statistical tests used in side channel cryptanalysis, e.g. differential power analysis (DPA), do not even require precise implementation details, such as timing behavior and power leakage models, in order to extract secret keys. In general, once side channel traces have been collected from a target device this data is usually post processed and evaluated by some statistical analysis. Given two or more collections of samples, statistical significance tests evaluate the probability that these samples belong to the same parent group. In this work, several different side channel attacks and corresponding countermeasures, such as randomized masking techniques for software and hardware implementations, are discussed and evaluated.

### 1.2. Summary of Research Contribution

In Chapter 2, we begin with an overview of various statistical methods which have been applied in side channel cryptanalysis in the past. We briefly discuss simple power
1.2 Summary of Research Contribution

analysis (SPA) in Chapter 2.1, differential power analysis (DPA) in Chapter 2.2, higher order DPA in Chapter 2.3, alternative statistical hypothesis tests, such as the T-test in Chapter 2.4 and the correlation coefficient in Chapter 2.5, and finally template attacks based on multivariate signal analysis in Chapter 2.7.

In Chapter 3 we propose a new class of side channel attacks which exploit internal collisions in ciphers. In cryptography, the term collision is traditionally associated with the mapping of two different inputs to an equal output by some non-injective function, e.g. a hash function. We show that internal collisions which do not propogate until the final output of a cipher can be detected with side channel cryptanalysis. In Chapter 3.2, we propose an internal collision attack against the Data Encryption Standard (DES). We exploit the fact that collisions can occur at the output of three adjacent S-boxes. Knowledge of the corresponding cipher inputs makes it possible to determine possible key candidates. For example, we are able to achieve an internal collision at the output of an S-box triplet with an average minimum of 140 encryptions which reveals 10.2 key bits. As a proof of concept we applied the attack on a DES software implementation running on a 8051 microcontroller. this attack. In Chapter 3.3, we exploit partial collisions at the output of the MixColumns transformation used in the Advanced Encryption Standard (AES). We develop several variants and refinements of the attack. By taking advantage of the birthday paradox we show that it is possible to cause and detect a collision in an output byte of the MixColumns transformation with as little as 20 measurements. Furthermore, we show propose an optimized attack which detects collisions in all four output bytes with only 31 measurements which entirely reveals the corresponding 32-bit subkey. The attack can also be applied to all four columns in AES in parallel which makes it possible to extract an entire 128-bit round key with only 40 encryptions. As in the case of DES, we perform the attack against a software assembly implementation of AES and discuss our results. Finally, in Chapters 3.4 and 3.5 we present two further applications of internal collision attacks against the block ciphers Serpent and Kasumi, respectively.

In Chapter 4, we propose various masking techniques used to protect AES software implementations against side channel attacks, such as first and higher order DPA and the aforementioned internal collision attacks. The target device mainly used in this chapter for validating the proposed countermeasures is the Atmel AVR microcontroller, however, other typical microprocessors, such as the 8051, could have been chosen, as well. In Chapter 4.3, an efficient masking scheme for AES software implementations based on inversions in the composite field is presented. We present the performance figures of various AES assembly implementations which use this countermeasure to thwart first order side channel attacks. We give a theoretical proof of security for the proposed scheme and validate its security with actually performed DPA attacks. In Chapter 4.4, we discuss the theoretical background of higher order side channel attacks and propose various masking strategies which lead to higher order resistant AES software implementations. We derive the theoretical correlation coefficients for DPA attacks of an arbitrary order
— both for the Hamming weight power leakage model and a more general leakage model. This makes it possible for implementers to assess the costs in terms of measurements for successful DPA attacks. We show that the measurement costs increase exponentially as the order of the DPA attack increases. We present the performance figures of AES assembly implementations which are resistant against DPA attacks of various orders.

In Chapter 5, we discuss two novel applications of template attacks which are based on multivariate signal classification as originally published by Chari et al. in 2002 [CRR02]. In Chapter 5.2 we show that a single side channel trace contains enough information to reveal the state of a single bit. Based on this single-bit classification, a new attack is presented in Chapter 5.3, which can defeat the masking countermeasure [GP99, AG01] under certain conditions. This presumes that the adversary has access to a smart card or cryptographic token which is protected by the masking countermeasure, but whose random number generator is defective or has some imperfect bias. The adversary can then build templates which allow to break an identical smart card even if its RNG is perfectly functional and has no bias. Thus, this attack opens up a potential back door by giving smart card manufacturers, vendors and developers the possibility to break their own implementations, if they have access to a card, whose RNG is biased, bust or has been intentionally destroyed. As a proof of concept, we show in Section 5.3.3 the results of attacks against a masked implementation of the DES running on a 6805-based smart card a masked implementation of the AES running on an AVR-based smart card.

In Chapter 6, we discuss the vulnerabilities of a masked AES hardware implementation realized in CMOS logic. In Chapter 6.2 we present the results of three different attacks, i.e. zero-offset DPA, toggle-count DPA and zero-input DPA. Comparison of the three attacks makes clear that toggle-count and zero-input DPA require far less measurements than zero-offset DPA. We show that these two attacks work so well due to the occurrence of glitches in masked circuits. Motivated by this fact, we show in Chapter 6.3 which parts of masked AES S-boxes prevent the propagation of glitches for certain inputs and thus result in a side channel vulnerability. The analysis reveals that the propagation of glitches is significantly influenced by the switching characteristics of XOR gates in masked multipliers. Masked multipliers are the basic building blocks of most recent proposals for masked AES S-boxes based on inversions in the composite field. We subsequently show in Chapter 6.4 that the side channel leakage of masked multipliers can be prevented by enforcing timing constraints for the XOR gates in each multiplier of an AES S-box. We also briefly present two approaches which show how these timing constraints can be realized in practice.
2. Overview of Side Channel Cryptanalysis

This chapter provides a brief introduction to side channel cryptanalysis and related topics. In cryptography the term side channel describes an unintentional physical source which leaks key-related information during the execution of a cipher. As a matter of fact, side channel cryptanalysis makes clear that it is not sufficient to design ciphers, which are merely secure from a mathematical point of view. Equally important is a careful and secure implementation of ciphers by skilled engineers.

![Diagram of cipher system]

Figure 2.1.: Side channel leakage of a cryptographic implementation.

In the past, several side channels have been investigated and publicly discussed. Typical side channels are the execution time of a cipher [Koc96], its power consumption [KJJ99, MDS99, MS00] and its electromagnetic radiation [GMO01, AARR02]. Even the sonic emissions of capacitances in a Personal Computer (PC) were analyzed in order to successfully exploit an e-mail encryption software [ST04]. Even though all these works first appeared in the mid or late 1990’s the partial declassification of the Transient Electromagnetic Pulse Emanation Standard (TEMPEST) documents by the U.S. National Security Agency (NSA) made clear that similar methods have been known and used in international espionage at least since the end of world war two [Joe00].

Side channel attacks belong to the more generic class of implementation attacks. As the name suggests these attacks aim at actively manipulating or tampering with cryptographic implementations, e.g. by inserting faults at particular points in time [ABF02], passively observing the behavior of a cryptographic implementation, e.g. in form of side
channel attacks [Koc96, KJJ99, MDS99], or semi-passively monitoring and evaluating
internal signals in an implementation, e.g. the data bus of a smart card used in banking
applications [ISW03].

This thesis deals with various topics of side channel cryptanalysis. Passively monitoring
the power consumption or electromagnetic radiation of a cryptographic device has some
significant advantages over other implementation attacks: physical access to the target
device can be restricted (e.g. when recording electromagnetic traces with an antenna
from another room). Moreover, some statistical test used in side channel cryptanalysis,
e.g. differential power analysis (DPA), do not even require precise implementation de-
tails, such as timing behavior and power leakage models, in order to extract secret keys.
In general, once side channel traces have been collected from a target device this data
is usually post processed and evaluated by some statistical analysis. Given two or more
collections of samples, statistical significance tests evaluate the probability that these
samples belong to the same parent group. The following sections review methods and
statistical tests commonly used in side channel attacks.

2.1. Simple Power Analysis

Simple Power Analysis (SPA) was introduced by Kocher et al. in 1998 in [KJJ98]. Since
then SPA has been extensively discussed in the scientific literature [KJJ99, CJR+99a,
MDS99, MS00, ABDM00, MS00, CKN00, QK02, LPW05]. In their original technical
report [KJJ98], Kocher et al. demonstrated that they were able to break the Data
Encryption Standard (DES) running on various smart cards by analysis of the power
consumption. In general, any ciphers implemented as software and executed by a micro-
processor continuously changes the state of internal registers, busses and memory cells.
Equally, hardware implementations of ciphers change the state of internal signal lines,
buffers, and so on.

Both microprocessors and Application Specific Integrated Circuits (ASICs) are com-
monly built in Complementary Metal Oxide Semiconductor (CMOS) logic [JM97]. In
Figure 2.2, the basic-building block of all CMOS circuits, i.e. the CMOS inverter, which
consists of a P-channel Metal Oxide Semiconductor (PMOS) and an N-channel Metal
Oxide Semiconductor (NMOS) transistor is shown. During idle operation the voltage
levels of the input and output node will be either at $V_{dd}$ (high) or $V_{ss}$ (low) and the
static dissipation is negligible\(^1\). However, during a transition of the input node from
$V_{dd}$ to $V_{ss}$ or vice versa a short circuit current will flow from $V_{dd}$ to $V_{ss}$ through the
two transistors [Men98, CKN00]. Moreover, load capacitances, e.g. busses and driven
gates, are charged or discharged at this time. As an approximate rule of thumb this

\(^{1}\text{This assumption ignores drift currents, which can range from 10 nA to 10 } \mu \text{A depending on the dimensions of the transistors [WE93, CKN00].} \)
latter dynamic dissipation typically accounts for up to 80% of the power dissipation of a device, while 15% are due to its short circuit dissipation and 5% are due to static dissipation [CKN00, QK02].

The power consumption of a microprocessor can be observed by measuring the voltage drop over a small shunt resistance $R_s$ (e.g. a few 10 Ω) between the $V_{ss}$ pin of the microprocessor and external ground of the power supply. Another possibility is the usage of an inductive current probe, which measures current induced by the power line, but has a lower bandwidth. A digital oscilloscope with low quantization noise (e.g. a 12-bit analog/digital converter) and high sampling rate (e.g. 500 MS/s) is typically used to digitize the current. This setup is shown in Figure 2.3.

In SPA attacks it is presumed that an adversary knows characteristic details of the device under test such as its power consumption characteristics and the time when key-related leakages occur. For example, Messerges et al. observed in [MDS99] that the number of bits of a bus or an internal register, which flip during a clock edge, is linear proportional to the current absorption for some devices. Hence, the power consumption at time $t_0$ can
be modelled as

\[ I(t_0) = \epsilon \cdot W(P \oplus X) + N \]

where \( \epsilon \) denotes a hardware-dependent constant of proportionality, \( W(P \oplus X) \) denotes the Hamming distance between a previous state \( P \) and a new state \( X \) of a bus or an internal register which leaks at time \( t_0 \) and \( N \) denotes uncorrelated Gaussian noise\(^2\). In Figure 2.4, the average power consumption of an 8051-compatible microcontroller is plotted over the Hamming weight of an 8-bit operand fetched over the internal data bus. The plot confirms an approximate linear proportionality between power consumption and number of bit transitions in the device.

![Figure 2.4: Hamming weight leakage of a microprocessor.](image)

In [MDS99, Müh01], Messerges et al. published an SPA attack against the key schedule algorithm in a DES implementation. This attack focuses on leakages during the PC2 permutations, derives information about the round keys and, effectively reduces the costs of an exhaustive key search from \( 2^{56} \) to \( 2^{38} \) encryptions. However, as discussed by Akkar et al. in [ABDM00], depending on the target device the leakage of internal registers and data or address busses is often not linear, and, hence, better and more complex power leakage models, e.g. the stochastic model proposed by Schindler and Lemke [SLP05], are required.

\(^2\)\( N \) represents the sum of all noise sources, i.e. intrinsic noise of the device under test, quantization noise of the oscilloscope, noise induced by the power source, etc. The central limit theorem [BSG+96, Müh00] states that \( N \) has an approximate Gaussian distribution.
2.2 Differential Power Analysis

As discussed in [MDS99], in software implementations of ciphers carry flag related instructions on key data, e.g. key byte shift operations or the use of conditional branches, may also result in SPA vulnerabilities. String or memory comparison instructions typically perform a conditional branch when a mismatch occurs. Hence, in order to thwart SPA attacks implementers may have to avoid or blind key-dependent branches and single-bit instructions which process key bits.

2.2. Differential Power Analysis

Similar to SPA, standard DPA also focuses on side channel measurements at a particular instance of time. However, in DPA multiple side channel measurements are partitioned into two sets depending on the boolean state of a key-dependent intermediate variable, which is predicted by an adversary using known plaintext/ciphertext and a key hypothesis. The biggest advantage of DPA over SPA attacks is the fact that neither side channel leakage models nor timing characteristics of the target device must be known. In order to protect cryptographic implementations various side channel countermeasures based on hardware and software have been discussed in the past [CJR+99b, GP99, CC00, Sha00]. For example, splitting intermediate, key-dependent variables into a number of statistically independent shares is commonly used by smart card developers to protect software implementations against standard DPA attacks. Higher-order DPA tries to defeat masking techniques by correlating multiple shares with joint statistical methods [Mes00b, AG03]. However, it was shown in [CJR+99b, SP06] that the measurement costs of higher-order DPA attacks increase exponentially with the number of shares, thus, often rendering higher-order DPA impractical.

DPA is a simple statistical hypothesis test, which evaluates the difference of means: an adversary makes a key hypothesis and partitions measured side channel traces into two sets depending on the outcome of a selected boolean function of the key hypothesis and known plaintexts or ciphertexts. Then, he/she computes the difference of means, i.e. a differential trace, of the two sets for each key hypothesis. If a resulting differential trace exhibits distinct peaks, the corresponding key hypothesis is assumed to be correct. In general, the selected function predicts a key-dependent intermediate bit occurring within the cipher. It is also possible to predict the state of a multi-bit variable in a cipher. This approach is casually called multi-bit DPA. Despite its name multi-bit DPA really requires more advanced statistical hypothesis tests, such as the correlation coefficient (see Section 2.6).

As an example, let us investigate a typical DPA attack against the block cipher DES. First, an adversary generates (or observes) $M$ random plaintexts $X_i$ and acquires the power traces $I_j(t)$ of the corresponding encryptions. For each plaintext $X_i$, the adversary is able to predict the state $D(X_i, K_h)$ of a chosen S-box output bit in round one based on a key hypothesis $K_h$. Since an S-box output in round one depends on six key bits, the
adversary has to make a 6-bit hypothesis $K_h$ in order to predict the input and output of the chosen S-box (see Figure 2.5).

![Figure 2.5: Input and output of a DES S-box.](image)

Depending on the outcome of the boolean selection function $D(X_i, K_h)$, side channel traces are assigned to a 0-partition or to a 1-partition. The difference of means of both partitions results in a differential trace $\Delta_{K_h}(t)$, which contains deviations from zero for the correct key hypothesis, if the power consumption is related to the state of the predicted bit. Moreover, these deviations from zero occur at those times when the predicted bit is processed by the device. DPA is defined as the difference of empirical means.

\[
\Delta_{K_h}(t) = \mu_{1,K_h}(t) - \mu_{0,K_h}(t) = \frac{\sum_{i=1}^{M} D(X_i, K_h) \cdot I_i(t)}{\sum_{i=1}^{M} D(X_i, K_h)} - \frac{\sum_{i=1}^{M} (1 - D(X_i, K_h)) \cdot I_i(t)}{\sum_{i=1}^{M} (1 - D(X_i, K_h))}
\]

Hence, after having measured $M$ side channel traces, the adversary needs to compute $|K_h|$ (in this example $2^6 = 64$) differential traces $\Delta_{K_h}(t)$ and decide in favor of the key hypothesis $K_h$, for which $|\Delta_{K_h}(t)|$ has the largest peaks among all differential traces. Let us assume that the Hamming weight of the predicted 4-bit S-box output leaks at time $t_0$, i.e.,

\[
I(t_0) = \epsilon \cdot W(S(X_i \oplus K)) + N
\]

where $N$ denotes additive Gaussian noise with mean $\mu_N$ and variance $\sigma^2_N$. If the plaintexts $X_i$ are uniformly distributed, the corresponding S-box output bits in round one will be uniformly distributed, as well, due to the design criteria of the DES S-boxes [Cop94], i.e. $M_{1,K_h} = \sum_{i=1}^{M} D(X_i, K_h) \approx \frac{M}{2}$ and $M_{0,K_h} = \sum_{i=1}^{M} 1 - D(X_i, K_h) \approx \frac{M}{2}$. Then, the expected value and the variance of this Hamming weight leakage are

\[
E[I(t_0)] = \epsilon \cdot 2 + \mu_N \quad V[I(t_0)] = \epsilon^2 \cdot 1 + \sigma^2_N
\]
Based on the non-linearity of the S-boxes, an incorrect key hypothesis, i.e. $K_h \neq K$, results in a bifurcation of power traces into the 0-partition and 1-partition, which is uncorrelated with the hypothesized S-box output bit\(^3\). In this case, the incorrectly predicted S-box output bit as well as the remaining three S-box output bits are approximately uniformly distributed in both partitions.

\[
\lim_{M \to \infty} \mu_{1,K_h}(t_0) = \lim_{M \to \infty} \mu_{0,K_h}(t_0) = E[I(t_0)] = \epsilon \cdot 2 + \mu_N \tag{2.4}
\]

\[
\Rightarrow E[\Delta_{K_h}(t_0)] = \lim_{M \to \infty} \mu_{1,K_h}(t_0) - \mu_{0,K_h}(t_0) = 0 \tag{2.5}
\]

\[
\Rightarrow V[\Delta_{K_h}(t_0)] = V[\mu_{1,K_h}(t_0)] + V[\mu_{0,K_h}(t_0)]
= 2 \cdot \frac{2}{M} \cdot (\epsilon^2 \cdot 4 \cdot \frac{1}{4} + \sigma_N^2) = \frac{4 \cdot (\epsilon^2 + \sigma_N^2)}{M} \tag{2.6}
\]

If the adversary guesses the key $K$ correctly, i.e. $K_h = K$, the hypothesized S-box output bit is fixed in the 0-partition and in the 1-partition while three of the four S-box output bits are still approximately uniformly distributed.

\[
\lim_{M \to \infty} \mu_{1,K_h}(t_0) = \epsilon \cdot (1 + \frac{3}{2}) + \mu_N = \epsilon \cdot 2.5 + \mu_N \tag{2.7}
\]

\[
\lim_{M \to \infty} \mu_{0,K_h}(t_0) = \epsilon \cdot (0 + \frac{3}{2}) + \mu_N = \epsilon \cdot 1.5 + \mu_N \tag{2.8}
\]

\[
\Rightarrow E[\Delta_K(t_0)] = \lim_{M \to \infty} \mu_{1,K_h}(t_0) - \mu_{0,K_h}(t_0) = \epsilon \neq 0 \tag{2.9}
\]

\[
\Rightarrow V[\Delta_K(t_0)] = V[\mu_{1,K_h}(t_0)] + V[\mu_{0,K_h}(t_0)]
= 2 \cdot \frac{2}{M} \cdot (\epsilon^2 \cdot 3 \cdot \frac{1}{4} + \sigma_N^2) = \frac{4 \cdot (\epsilon^2 \cdot \frac{3}{4} + \sigma_N^2)}{M} \tag{2.10}
\]

Therefore, if the adversary guesses the key correctly, the corresponding differential trace $\Delta_K(t)$ will feature a peak of height $\epsilon$ at time $t_0$. The variance of the differential trace decreases, i.e. the "quality" of a DPA attack increases, as the number of measurements $M$ increases. As a result, every successful DPA attack against a DES S-box reduces the costs of an exhaustive key search by six bits. Since four boolean selection functions are possible for each DES S-box output, it is also possible to combine the four resulting differential traces, which further decreases measurement costs [BK02].

\(^3\)Please note that the S-boxes of DES are not perfectly non-linear. As a result, so-called "ghost peaks" have been reported, which occur in differential traces for false key hypotheses [BK02, BCO04]
2.3. Higher Order Differential Power Analysis

The notion of higher order DPA was already mentioned in Kocher et al.’s groundbreaking article [KJJ99]: ”Of particular importance are high-order DPA functions that combine multiple samples from within a trace“. Thus, the order of a DPA\(^4\) attack denotes the number of points that are combined within each trace by some joint function chosen by the adversary. Interestingly, higher order DPA attacks make it possible to break the so-called masking countermeasure which is commonly used to protect hardware and software implementations of ciphers against DPA attacks by blinding all intermediate key-dependent variables with one or more random masks [Mes00a, AG01].

For simplicity, let us focus on second order DPA\(^5\) in the subsequent text, because these attacks have been widely investigated in side channel cryptanalysis [Mes00b, WW04, JPS05, OMHT06]. Second order DPA is based on the following fact: even if a random mask and the corresponding masked variable are statistically independent, their joint distribution is correlated with the unmasked variable which is predictable by the adversary. However, higher order attacks also have a major drawback: they generally presume that the adversary knows the exact times when masks and masked data leak through the observed side channel. Let us assume the Hamming weight of an \(n\) bit mask \(R\) leaks at time \(t_0\) and the Hamming weight of the masked \(n\) bit variable \(Y = X \oplus R\) leaks at time \(t_1\).

\[
I(t_0) = \epsilon \cdot W(R) + N_0 \quad I(t_1) = \epsilon \cdot W(Y) + N_1
\]

where \(N_0, N_1\) denote additive Gaussian noise with means \(\mu_0, \mu_1\) and variances \(\sigma_0^2, \sigma_1^2\), respectively. Let \(X\) be a key-dependent variable, e.g. an S-box output, which is hypothesesizable by the adversary. Let an adversary predict the least significant output bit of an S-box, i.e. \(X[0]\). If \(X[0] = 0\), then \(Y[0] = R[0]\) and the expected value of the joint product \(I(t_0) \cdot I(t_1)\) is

\[
\mathbb{E}_{X[0]=0} = E[I(t_0) \cdot I(t_1)] = \frac{1}{2} \cdot \left( \epsilon^2 \cdot \left( \frac{n-1}{2} \right)^2 + \epsilon \cdot \frac{n}{2} \cdot (\mu_0 + \mu_1) + \mu_0 \cdot \mu_1 \right)
+ \frac{1}{2} \cdot \left( \epsilon^2 \cdot \left( 1 + \frac{n-1}{2} \right)^2 + \epsilon \cdot \frac{n}{2} \cdot (\mu_0 + \mu_1) + \mu_0 \cdot \mu_1 \right)
= \epsilon^2 \cdot \left( \frac{n}{2} + \left( \frac{n-1}{2} \right)^2 \right) + \left( \epsilon \cdot \frac{n}{2} \cdot (\mu_0 + \mu_1) + \mu_0 \cdot \mu_1 \right)
\]

\(^4\)or any other statistical hypothesis tests
\(^5\)General higher order DPA attacks are extensively discussed in Chapter 4.4.
If \( X[0] = 1 \), then \( Y[0] \neq R[0] \) and the expected value of the joint product \( I(t_0) \cdot I(t_1) \) is

\[
\overline{T}_{X[0]=1} = E[I(t_0) \cdot I(t_1)] = \epsilon^2 \cdot \left( \frac{n - 1}{2} + \left( \frac{n - 1}{2} \right)^2 \right) + \left( \epsilon \cdot \frac{n}{2} \cdot (\mu_0 + \mu_1) + \mu_0 \cdot \mu_1 \right)
\]

(2.13)

For the correct key hypothesis the differential trace is

\[
\lim_{M \to \infty} \Delta_{K_h} = \overline{T}_{X[0]=1} - \overline{T}_{X[0]=0} = -\frac{\epsilon^2}{2}
\]

(2.14)

while the differential traces of false key hypotheses approximate zero with an increasing number of measurements. It is also possible to use other joint functions in the preprocessing step. For example, in [OMHT06] Oswald et al. show that the absolute difference \( |I(t_0) - I(t_1)| \) is advantageous, if the side channel leaks the Hamming weight. Please note, that DPA attacks of an arbitrary order are also possible, if the target device does not leak the Hamming weights of processed data. This is discussed in more detail in Chapter 4.4.4.

### 2.4. T-Test

As proposed in [CKN00, Mühl00, BK02, AO], a statistical hypothesis test which also considers the variance of the 0-partition and the 1-partition is known as the T-Test. Figure 2.6 clarifies the meaning of the T-Test. In the left plot two distributions with means \(-c\) and \(c\) and standard deviation \(\sigma\) are shown. In the right plot two additional distributions with means \(-c\) and \(c\) but standard deviation \(\frac{1}{4} \cdot \sigma\) are shown. It is obvious that the distributions in the left plot are more similar, i.e. there is more overlap than in the distributions of the right plot. However, a difference of means test (DPA) would result in equal differential traces. On the other hand the T-Test normalizes the
differential trace by dividing by its standard deviation. The T-Test is defined as

\[
T_{Kh}(t) = \frac{\Delta_{Kh}(t)}{\sqrt{\frac{\sigma^2_{0,Kh}(t)}{M_{0,Kh}} + \frac{\sigma^2_{1,Kh}(t)}{M_{1,Kh}}}} = \frac{\mu_{1,Kh}(t) - \mu_{0,Kh}(t)}{\sqrt{\frac{\sigma^2_{0,Kh}(t)}{M_{0,Kh}} + \frac{\sigma^2_{1,Kh}(t)}{M_{1,Kh}}}}
\]

(2.15)

where \(M_{0,Kh}\) and \(M_{1,Kh}\) denote the number of signals in the 0-partition and 1-partition, and \(\sigma^2_{0,Kh}(t)\) and \(\sigma^2_{1,Kh}(t)\) denote the variances of the signals in the 0-partition and 1-partition, respectively. Due to the fact that noise parts often have a greater standard deviation than signal parts in a power trace, the T-Test is a good method to reduce these peaks in a differential trace [BK02].

2.5. Void Hypothesis DPA

Another improved hypothesis test which is based on the difference of means but also involves the signal variances was proposed by Agrawal et al. in [ARR03]. This test uses a void key hypothesis \(K_v\) which corresponds to a random bifurcation of power traces into a 0-partition and 1-partition. Let the 0-partition and 1-partition of the void hypothesis contain \(M_{0,K_v}\) and \(M_{1,K_v}\) power traces, with \(M_{0,K_v} \approx M_{1,K_v} \approx \frac{M}{2}\). Then, these partitions have the empirical means \(\mu_{0,K_v}(t)\), \(\mu_{1,K_v}(t)\), and variances \(\sigma^2_{0,K_v}(t)\) and \(\sigma^2_{1,K_v}(t)\). If the difference of means of the 0-partition and 1-partition of the void hypothesis is denoted by \(\Delta_{K_v}(t) = \mu_{1,K_v}(t) - \mu_{0,K_v}(t)\), the void hypothesis test is defined as

\[
M_{Kh}(t) = \frac{(\Delta_{Kh}(t) - \Delta_{K_v}(t))^2}{\frac{\sigma^2_{0,Kh}(t)}{M_{0,Kh}} + \frac{\sigma^2_{1,Kh}(t)}{M_{1,Kh}}} - \ln \left(\frac{\sigma^2_{0,Kh}(t)}{M_{0,Kh}} + \frac{\sigma^2_{1,Kh}(t)}{M_{1,Kh}}\right)
\]

(2.16)

As shown in [ARR03], the void hypothesis test performs much better than standard DPA, i.e. the correct key hypothesis can be distinguished from incorrect key hypotheses with less measurement costs, because it uses a squared metric and takes signal variances into account. The main disadvantage of this test is based on the fact that its results are not reproducible, because the random bifurcation of side channel traces for the void hypothesis is probabilistic.

2.6. Correlation Coefficient

In statistics, a common method to measure the linear relationship between two random variables, e.g. \(X\) and \(Y\), is the correlation coefficient [Mih00, LSP04, AO]. For example, the Walsh coefficient used in linear cryptanalysis is closely related to the more generic
correlation coefficient \cite{dob01}. It is defined as

\[ \rho(X, Y) = \frac{COV[X, Y]}{\sqrt{V[X]} \cdot \sqrt{V[Y]}} = \frac{E[X \cdot Y] - E[X] \cdot E[Y]}{\sqrt{V[X]} \cdot \sqrt{V[Y]}} \]  

(2.17)

The correlation coefficient provides a normalized measure: if \( \rho(X, Y) = -1 \), \( X \) and \( Y \) are perfectly anti-correlated, if \( \rho(X, Y) = 0 \), \( X \) and \( Y \) are perfectly uncorrelated, and, if \( \rho(X, Y) = 1 \), \( X \) and \( Y \) are perfectly correlated. Due to its normalized output it is easy to compare the results of different side channel attacks. As stated in Section 2.2, in many devices the Hamming weight of a processed variable leaks through a side channel. Picking up our previous example of an attack against DES, let the selection function \( D(X_i, K_h) \) denote the Hamming weight of a 4-bit output of a chosen S-box in round one for a known plaintext \( X_i \) and hypothesized key \( K_h \). The empirical correlation coefficient turns out to be

\[ r(I_i(t), D(X_i, K_h)) = \frac{\sum_{i=1}^{M} I_i(t) \cdot D(X_i, K_h)}{\sqrt{\sum_{i=1}^{M} (I_i(t) - \bar{I}_i(t))^2} \cdot \sum_{i=1}^{M} (D(X_i, K_h) - \bar{D}(X_i, K_h))^2} - \frac{1}{M} \cdot \sum_{i=1}^{M} I_i(t) \cdot \sum_{i=1}^{M} D(X_i, K_h) \]  

\[ \sqrt{\sum_{i=1}^{M} (I_i(t) - \bar{I}_i(t))^2} \cdot \sum_{i=1}^{M} (D(X_i, K_h) - \bar{D}(X_i, K_h))^2} \]  

(2.18)

where \( \bar{I}_i(t) \) denotes the average power consumption and \( \bar{D}(X_i, K_h) \) denotes the average outcome of the selection function, i.e. in the DES S-box output example \( \bar{D}(X_i, K_h) \approx 2 \). The empirical correlation coefficient \( r(I_i(t), D(X_i, K_h)) \) converges against the theoretical correlation coefficient \( \rho(I_i(t), D(X_i, K_h)) \) with an increasing number of measurements.

\[ \rho(I_i(t), D(X_i, K_h)) = \lim_{M \to \infty} r(I_i(t), D(X_i, K_h)) \]  

(2.19)

During the implementation of a cryptographic algorithm it is often possible for the designers to derive accurate power/EM leakage models and, thus, assess the theoretical correlation coefficients\(^6\) of side channel attacks \cite{man04}. In order to estimate the number of measurements required for a successful attack, a statistical confidence interval based on the Fisher Z-transformation of the theoretical correlation coefficient has to be used \cite{muh00}. Essentially, the confidence interval expresses the likelihood that an empirical correlation coefficient of an incorrect key hypothesis is accepted by the hypothesis test as the correct key hypothesis for a given number of measurements \cite{spr05}.

Let us assume an adversary repeats several side channel attacks under the same conditions, i.e. with equal number of measurements \( M \) and exactly the same measurement

\(^6\)For example, in Sections 4.4 and 6.2 the theoretical correlation coefficients of various attacks are derived for the Hamming weight leakage model.
setup parameters. An analysis of the empiric correlation coefficients \( r(I_i(t), D(X_i, K_h)) \) for a fixed key hypothesis reveals that their mean converges the theoretical correlation coefficient \( \rho(I_i(t), D(X_i, K_h)) \), however, it would also reveal that the empiric correlation coefficients are not normally distributed. In order to use confidence intervals, it is necessary to first transform the empiric correlation coefficients to normally distributed random variables with the Fisher Z-transformation [Man04, SPRQ05, OMHT06].

\[
\begin{align*}
  z &= \frac{1}{2} \cdot \ln \left( \frac{1+r}{1-r} \right) \\
  \mu_z &= \frac{1}{2} \cdot \ln \left( \frac{1+\rho}{1-\rho} \right) \\
  \sigma_z^2 &= \frac{1}{M-3} \approx \frac{1}{M} \quad \text{if} \quad M >> 1
\end{align*}
\]

Hence, Z-transformed empiric correlation coefficients are normally distributed with mean \( \mu_z \) and variance \( \sigma_z^2 = \frac{1}{M-3} \). The means of Z-transformed empiric correlation coefficients for correct and incorrect key hypotheses are

\[
\begin{align*}
  \mu_{z,K=K_h} &= \frac{1}{2} \cdot \ln \left( \frac{1+\rho}{1-\rho} \right) \\
  \mu_{z,K\neq K_h} &= \frac{1}{2} \cdot \ln \left( \frac{1+0}{1-0} \right) = 0
\end{align*}
\]

In order to derive the probability that the Z-transformed empiric correlation coefficient \( z_{K=K_h} \) of the correct key guess is greater than a Z-transformed empiric correlation coefficient \( z_{K\neq K_h} \) of a wrong key guess, we need to evaluate their distance, i.e. \( d = z_{K=K_h} - z_{K\neq K_h} \). As explained in [Ash93], the distance \( d \) is also normally distributed with mean \( \mu_{d} = \frac{1}{2} \cdot \ln \left( \frac{1+\rho}{1-\rho} \right) \) and variance \( \sigma_d^2 = \frac{2}{M-3} \).

\[
P(z_{K=K_h} > z_{K\neq K_h}) = P(d = z_{K=K_h} - z_{K\neq K_h} > 0) = 1 - P(d = z_{K=K_h} - z_{K\neq K_h} < 0) = \Phi \left( \frac{1}{2} \cdot \ln \left( \frac{1+\rho}{1-\rho} \right) \right)
\]

Solving this equation for the required number of measurements \( M \) results in:

\[
M = 3 + 8 \cdot \left( \frac{\Phi^{-1}(P(d > 0))}{\ln \left( \frac{1+\rho}{1-\rho} \right)} \right)^2
\]

As suggested in [Man04], a conservative confidence interval is \( P(d > 0) = 0.9999 \), i.e. \( \Phi^{-1}(P(d > 0)) \approx 3.719 \), which basically states that 99.99% of all correlation coefficients of wrong key guesses shall be less than the correlation coefficient of the correct key guess.
2.7. Template Attacks

Classical SPA and DPA evaluate the side channel leakage of key-related information univariately, i.e., at a particular instance of time [KJJ99, MDS99, CCD00, CJR+99b, CJR+99a]. This neglects the fact that this information, or information related to it, may also leak at several additional points in time in a side channel trace. For example, in an AES software implementation an S-box output in round one may leak during the fetch instruction, which reads the S-box output from a look-up table, but also up to four times during the execution of the subsequent MixColumns transformation. In [CRR02], Chari et al. showed that noise levels contained in a side channel trace may significantly conceal underlying signal levels. As discussed in Section 2.2, the variance of noise in an averaged signal decreases anti-proportionally as the number of traces increases. Nevertheless, in situations where only a limited number of side channel traces is available, e.g. if the number of encryptions for a fixed key are bounded by a protocol, univariate signal analysis used in SPA and DPA is sub-optimal.

Another class of side channel attacks, which is closely related to SPA, consists of two phases: a training phase and an classification phase. During the training phase power consumption or EM emission models, so-called templates, are derived from a test device and it is assumed that an adversary has full access to this device. During the classification phase these templates are used to recognize the state of key-related intermediate variables in a side channel trace derived from an identical target device with restricted access. This approach was initially suggested by Faun et Pearson in [FP99] and later significantly elaborated by Chari et al. in [CRR02]. The template attack proposed by Chari et al. is more eminent, because it is based on multivariate statistics and maximum likelihood hypothesis testing [ARR03]. Multivariate signal analysis considers the side channel leakage at several points in time and the joint distributions of these leakages. The notion of multivariate signal analysis originates from the statistician Thomas Bayes who lived in the first half of the eighteenth century [Hof98].

Hence, a template describes the statistical relationship of those points in a side channel trace, which strongly correlate with a key-dependent intermediate variable, e.g., an S-box output. Every template consists of a vector of means of these significant points and a matrix, whose elements represent the covariances of all possible pairs of significant points. If the adversary has collected $M$ side channel traces $I_j(t)$ from the test device during the training phase, the noise-free signal $\overline{I(t)}$ can be estimated by averaging these $M$ traces.

$$
\overline{I(t)} = \frac{1}{M} \cdot \sum_{j=1}^{M} I_j(t)
$$

(2.29)

In order to create a noise model for each state of the observed key-dependent intermediate
variable, the first step is to compute noise vectors $N_j(t)$ for every trace by subtraction of the estimated noise-free signal.

\[ N_j(t) = \overline{I(t)} - I_j(t) \quad 1 \leq j \leq M \tag{2.30} \]

The number of significant points per averaged power trace $\overline{I(t)}$ and noise vectors $N_j(t)$ is then usually reduced to a smaller number $L$ (e.g. 20-50) in order to reduce processing power during the subsequent classification phase. The selection of significant points was discussed in detail by Rechberger et Oswald in [RO04]. The matrix $C$ contains all estimated pairwise covariances of significant points.

\[
C(t_x, t_y) = \text{cov}(I_j(t_x), I_j(t_y)) \quad 1 \leq x, y \leq L \\
= \frac{1}{M} \cdot \sum_{j=1}^{M} \left( I_j(t_x) - \overline{I_j(t_x)} \right) \cdot \left( I_j(t_y) - \overline{I_j(t_y)} \right) \\
= \frac{1}{M} \cdot \sum_{j=1}^{M} N_j(t_x) \cdot N_j(t_y) \tag{2.31}
\]

Matrix $C$ is symmetric and its diagonal elements $C(t_x, t_x)$ represent the variances of significant points. Let us assume the key-dependent variable can have $S$ states, e.g., $S = 256$ for an AES S-Box output. Once $S$ templates have been derived from the test device an adversary can mount an attack by measuring a trace $I(t)$ from an identical target device and examine which of the $S$ templates matches the noise characteristic of $I(t)$ best. Hence, during the classification phase the adversary first computes a noise vector $N'_i(t) = \overline{I_i(t)} - I'_i(t)$ for each template $1 \leq i \leq S$. Then, the probability that noise vector $N'_i(t)$ occurs for template $i$ can be evaluated with the multivariate Gaussian probability density function.

\[
p(N'_i) = \frac{1}{\sqrt{(2\pi)^L \cdot |C'_i|}} \cdot \exp \left( -\frac{1}{2} \cdot N'_i^T \cdot C'^{-1}_i \cdot N'_i \right) \tag{2.32}
\]

In [CRR02], Chari et al. prove that this approach of matching a corresponding template is optimal, if the significant points have a Gaussian distribution for a fixed template $1 \leq i \leq S$. Depending on the device this presumption holds true due to the central limit theorem.

The major advantage of template attacks over univariate SPA is the fact that classification rates are significantly higher [CRR02, ARR03, RO04, ARRS05]. As a result, a single side channel trace from the target device is often sufficient to extract a subkey. For example, Chari et al. state that they were able to successfully break implementations of RC4, DES and even an SSL accelerator card inside a closed server with a single electromagnetic trace [CRR02]. Hence, template attacks are especially suited to break stream ciphers or ciphers, which use ephemeral keys.
3. Internal Collision Attacks

In this chapter, a new class of side channel attacks is presented which tries to exploit internal key-dependent collisions in a cipher by analysis of the power consumption or the EM emission. These attacks are closely related to differential analysis [BS90] and are based on the fundamental hypothesis that an internal collision results in the execution of equal instructions and thus an equal side channel leakage.

In Chapter 3.1, we discuss the principle of internal collision attacks and the so-called birthday paradox. In Chapter 3.2, we propose an internal collision attack against the Data Encryption Standard (DES). Parts of this work were published in 2003 at the Fast Software Encryption (FSE) workshop [SWP03]. Furthermore, we also suggest a modified DES variant which is secure against internal collision attacks. Parts of this work were submitted to the Cryptographic Hardware and Embedded Systems (CHES) 2006 conference in [LPPS06]. In Chapter 3.3, we propose an internal collision attack against the Advanced Encryption Standard (DES). Parts of this work were published in 2004 at the Cryptographic Hardware and Embedded Systems (CHES) conference in [SLFP04]. In Chapter 3.4, we propose an internal collision attack against the block cipher Serpent. Finally, in Chapter 3.5, we propose an internal collision attack against the block cipher Kasumi.

3.1. Previous Work

In cryptography, the term collision denotes the mapping of different inputs to an equal output by a non-injective function. Typically, collisions are associated with hash functions which map an arbitrary input to an output of a fixed length. Let us assume a hash function \( h(M) \) maps an input \( M \) of arbitrary length to a fixed-length output value. A collision occurs, if \( l \geq 2 \) different inputs map to an equal output value.

\[
h(M_1) = h(M_2) = \ldots = h(M_l), \quad M_i \text{ pairwise different}, \quad l \geq 2
\]

Cryptanalysts have exploited collisions in hash functions for many years [dBB94, Vau94, Dob98, BGW98b]. Most of the previous attacks against hash functions only attacked a
few rounds, e.g., three rounds of RIPEMD [Dob97, NIS95], however in [Dob98], Dobbertin proposed a collision attack against the full round MD4 hash function [Riv92]. Dobbertin was able to show that MD4 is not collision-free and that collisions in MD4 can be found in a few seconds on a PC. Just recently, collision attacks against MD5, SHA-0 and SHA-1 were published [WY05, WYY05b, WYY05a]. Another historic example of breaking an entire hash function is the COMP128 algorithm [BGW98a]. COMP128 is widely used to authenticate mobile phones to base stations in Global System for Mobile Communication (GSM) networks [MC98]. The core building block of COMP128 is a hash function, which is based on a butterfly structure with five stages. In [BGW98b], it was shown that it is possible to cause a collision in the second stage of the hash function, which fully propagates to the output of the algorithm.

A generic attack against hash functions is based on the birthday paradox [Sta03, Buc04], which investigates the following question: how high is the probability $p$ that at least two out of $k$ people in a closed room have the same birthday. Let $q = 1 - p$ denote the probability that all $k$ persons have different birthdays, i.e.

$$ q = 1 - p = \frac{1}{n^k} \prod_{i=0}^{k-1} (n - 1) = \prod_{i=0}^{k-1} (1 - \frac{i}{n}) $$

where $n = 365$ denotes the number of days per year. Solving this equation for $q \leq 0.5$ by blind trial and error results in $k = 23$ persons. As a result, if 23 or more people are present, the probability that two birthdays match is $p \geq 0.5$. In order to analytically determine, whether $k$ fulfils the condition $q \leq 0.5$, the inequality $(1 - x) \leq e^{-x}, x \geq 0$ can be substituted into equation 3.1.

$$ q = 0.5 \leq \prod_{i=0}^{k-1} e^{-i/n} = e^{-\sum_{i=0}^{k-1} i/n} = e^{-k(k-1)/(2n)} $$

$$ \Rightarrow k(k-1) \leq 2n \cdot \ln(2) \quad \text{for } k \gg 1 \Rightarrow k^2 - k \approx k $$

$$ \Rightarrow k \leq \sqrt{2n \cdot \ln(2)} \approx 1.18 \cdot \sqrt{n} \approx \sqrt{n} $$

Thus, for $n = 365$ we get $k \approx 22.54$ which is a good approximation. In general, if a non-injective function, such as a hash function, maps inputs of arbitrary length to $n$-bit outputs, a collision occurs at the output after an average of $\sqrt{2n} = 2^{n/2}$ random trials. However, a brute force collision search against modern hash functions such as RIPEMD-160 [Dob97], which generate 160-bit hash outputs, takes approximately $2^{80}$ steps and, thus, is regarded infeasible in practice.

Most cryptographic algorithms such as block ciphers represent bijective functions\(^1\), i.e., collisions cannot occur at the output. However, \textit{internal collisions} may occur at some point within the cipher. Hence, we define the term \textit{internal collision} as a collision at the

\(^1\)if one of its inputs, i.e. the key or the plaintext/ciphertext is fixed
output of some sub-function within a cipher. As we will see in the succeeding sections, side channel analysis is an appropriate method to detect internal collisions. Furthermore, we will show that secret keys can be extracted from cryptographic implementations, if internal collisions at the output of a sub-function depend on secret key data and known plaintext/ciphertext. The idea to exploit internal collisions by analysis of side channel data originates from Hans Dobbertin [Dob02].

One approach to exploit internal collisions in generic Feistel ciphers was presented by Andreas Wiemers [Wie03]. This work was done independently from our work presented in Section 3.2. As shown in Figure 3.1, an adversary encrypts/decrypts the two inputs \((L, R)\) and \((L \oplus \Delta_L, R \oplus \Delta_R)\). By x-or addition of the differential \(\Delta_R\) to the right half of the plaintext/ciphertext and, thus, to the input of the embedded function \(f_k\) a key-dependent differential \(\Delta_{R'}\) occurs at its output. The adversary tries to compensate this output differential \(\Delta_{R'}\) by x-or adding the differential \(\Delta_L\) to the left half input of the Feistel cipher. If the differential \(\Delta_L\) equals the unknown output differential \(\Delta_{R'}\), a collision occurs at the output of the x-or addition.

\[
L \oplus f_k(R) = (L \oplus \Delta_L) \oplus (f_k(R) \oplus \Delta_{R'})
\]  

(3.5)

As a result, the same data will be processed in the embedded function \(f_k\) of the following Feistel round and, thus, the power consumption/EM radiation of the implementation will be equal during the execution of the following round, as well. If the side channel measurements indicate a collision, it is possible to derive secret key information by analysis of \((L, R)\) and \((L \oplus \Delta_L, R \oplus \Delta_R)\). In Section 3.5 we show an application of Wiemers’s attack against the Feistel cipher Kasumi.

![Figure 3.1.: Collisions in a Feistel cipher.](image)
3.2. Internal Collisions in DES

In this section, we show that it is possible to detect internal collisions at the output of three adjacent S-boxes of DES by side channel analysis. Since these collisions only occur for certain round inputs and round keys secret key information can thus be extracted. Beginning with Section 3.2.1, we show that collisions are not possible at the output of a single S-box due to the expansion permutation in the DES $f$-function. However, in Section 3.2.2, we show that it is possible to cause collisions in three adjacent S-boxes. This theory is refined and optimized in Section 3.2.3 where we show how to exploit an internal collision in the S-Box triplet 2, 3, 4 with an average\(^2\) number of 140 encryptions which reveals an average of 10.2 key bits. Internal collisions can also occur at the output of other S-box triplets, for example we show that collisions at the output of S-box triplet 7, 8, 1 are possible with an average number of 165 encryptions which reveal an average of 13.8 key bits. We successfully validate our attack by compromising an 8051 assembly implementation of DES in Section 3.2.4. In Section 3.2.5, we finally present a single optimized and cryptographically strengthened S-box as a possible countermeasure. The proposed S-box is more resistant against linear and differential cryptanalysis than the original eight S-boxes of DES and, furthermore, immune to internal collision attacks.

3.2.1. Collisions in a Single S-box

The Data Encryption Standard (DES) uses eight substitution tables, the so-called S-boxes [NIS77, Sti95, Sch96, MvOV97]. The S-boxes represent the cryptographic core of the DES and are based on eight non-linear\(^3\) mappings $S_i : 2^6 \rightarrow 2^4, i \in \{1, \ldots, 8\}$, which are non-injective and whose outputs are uniformly distributed. Due to the non-injectivity of the S-boxes, collisions can occur at the output of a single S-box. Because of the uniform distribution of the S-box output values, for each 6-bit input $z$ exist three different input values $z'$, $z''$ and $z'''$ and thus three input differentials $δ_1, δ_2, δ_3$, which result in an equal S-box output.

\[
S(z) = S(z') = S(z'') = S(z'''), \quad z \neq z' \neq z'' \neq z''' \quad (3.6)
\]

\[
S(z \oplus δ_1) = S(z \oplus δ_2) = S(z \oplus δ_3), \quad δ_1 \neq δ_2 \neq δ_3 \neq 0 \quad (3.7)
\]

A table can be generated for each S-box, which lists the inputs $z \in \{0, \ldots, 2^6 - 1\}$ for all occurring input differentials $δ_i \in \{1, \ldots, 2^6 - 1\}$, which exhibit a collision. In [Cop94],

\(^2\)averaged over 10,000 random keys

\(^3\)Please note that the non-linearity of the DES S-boxes is imperfect. This fact is exploited in the linear attack by Matsui [Mat94]. Moreover, this non-linearity gives rise to so-called ghost peaks in DPA attacks [BCO04], i.e., distinct peaks in differential traces with regard to false key hypotheses.
Coppersmith\(^4\) published some design criteria of the S-boxes. Criterion (S-7) states that no more than 8 pairs of inputs \((z, z \oplus \delta_i)\) can occur for each input differential \(\delta_i\).

In order to extract the six secret key bits \(k\) an adversary chooses a suitable\(^5\) \(\delta_i\) and randomly varies 6 bits of the plaintext input \(x = z \oplus k\) until he detects a collision 
\[
S(x \oplus k) = S(x \oplus k \oplus \delta_i).
\]

As shown in Figure 3.3, the two most and least significant bits of the inputs \(x\) and \(x \oplus \delta_i\) enter the adjacent S-boxes due to the bit spreading of the expansion permutation. The inputs of the adjacent S-boxes only remain unchanged, if the two most and least significant bits of differential \(\delta_i\) are zero. However, such a differential \(\delta_i\) does not exist due to S-box criterion (S-3) in [Cop94]. Criterion (S-3) states that each 4-bit S-box output occurs exactly once, if the least and most significant input bits are fixed. Therefore, a collision attack, which focusses on a single S-box, while preserving the inputs of the two adjacent S-boxes is not possible.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{collision_s_box}
\caption{Collisions in a single S-box of DES.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{bit_mask}
\caption{Bit mask for a collision in a single S-box.}
\end{figure}

### 3.2.2. Collisions in Three Adjacent S-boxes

As first mentioned by Davio et al. in [DDQ84] and later confirmed by Coppersmith in [Cop94], collisions at the output of two adjacent\(^6\) S-boxes are not possible, as well. However, it is possible to cause collisions at the output of three adjacent S-boxes, and,

\(^4\)Don Coppersmith was a member of the team of cryptographers at the IBM T.J. Watson Research Center, who designed the DES. However, it is widely believed, e.g. see [Sch96], that the National Security Agency (NSA) provided the final set of S-boxes which made DES secure against differential cryptanalysis.

\(^5\)Typically, the adversary chooses an input differential, which features a maximum collision probability at the S-box output. The upper bound of this probability is \(\frac{16}{64} = 0.25\), see S-box criterion (S-7).

\(^6\)S-boxes 8 and 1 are considered to be adjacent.
thus, at the output of the $f$-function. In this case the adversary needs to generates random 14-bit\(^7\) plaintext pairs $(x, x \oplus \Delta)$. The differential $\Delta = (\delta_j | \delta_{j+1} | \delta_{j+2})$ denotes the concatenation of three 6-bit input differentials $\delta_j, \delta_{j+1}, \delta_{j+2}$ with regard to S-boxes $j, j+1$ and $j+2$. As derived by Coppersmith in [Cop94], due to the S-box criteria the three differentials $\delta_j, \delta_{j+1}, \delta_{j+2}$ have to comply with the bit patterns

\[
\begin{align*}
\delta_j &= 00cd11|_2, \\
\delta_{j+1} &= 11gh10|_2, \\
\delta_{j+2} &= 10km00|_2
\end{align*}
\]

with $c,d,g,h,k,m \in \{0,1\}$. This bit pattern is shown in Figure 3.4. Hence, the theoretical maximum number of differentials $\Delta$ which result in collisions at the output of an S-box triplet is $2^6 = 64$. However, as stated in [Sch02], many differentials $\Delta$, which comply with the bit pattern stated above, do not exist for the S-boxes of DES. In general, an adversary is trying to minimize the costs of a collision search. Table 3.1 lists the maximum number out of $2^6 = 64$ S-box inputs for the optimum input differentials $\delta_j, \delta_{j+1}, \delta_{j+2}$ conform with conditions 3.8-3.10 with regard to all eight S-boxes [BB94]. Likewise, for each S-box triplet an optimum compound differential $\Delta_{opt}$ can be chosen, which results in collisions with maximum probability. These maximum probabilities $p_{max}$ and the corresponding number of 18-bit S-box triplet inputs $|Z_{\Delta_{opt}}|$ are listed in Table 3.2.

Collisions at the output of S-box triplets are for instance exploited by Biham and Shamir in differential cryptanalysis of DES [BS90], which gives rise to an attack with $2^{47.2}$ chosen plaintext messages. The designers of DES already suspected about differential cryptanalysis in the early 1970’s and, according to Coppersmith [Cop94], arranged the S-boxes in such an order as to minimize the probability of collisions at the output of S-box triplets. However, as stated in [BB94] by Biham, the order of S-boxes in DES is not optimum with regard to both differential and linear cryptanalysis, which appears to be an interesting contradiction.

\(^7\)We refer to $x$ and $x \oplus \Delta$ as the inputs of the $f$-function after propagation through the expansion box. In the remainder, $x$ and $x \oplus \Delta$ are described as 18-bit inputs, but due to the redundancy introduced by the expansion box the adversary only needs to vary 14 bits of the plaintext, i.e. $x, x \oplus \Delta \in \{0, \ldots, 2^{14} - 1\}$.
### 3.2 Internal Collisions in DES

<table>
<thead>
<tr>
<th>S-box</th>
<th>$00cd11_{12}$</th>
<th>$11gh10_{12}$</th>
<th>$10km00_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 3.1.: Maximum number of S-box Inputs which result in a collision at the output of a single S-box for input differentials $\delta_j, \delta_{j+1}, \delta_{j+2}$.

| S-boxes | $|Z_{\Delta_{opt}}|$ | $p_{max} = |Z_{\Delta_{opt}}| \cdot 2^{-18}$ |
|---------|----------------------|------------------------------------------|
| 1,2,3   | 1120                 | 0.0043                                   |
| 2,3,4   | 768                  | 0.0029                                   |
| 3,4,5   | 1024                 | 0.0039                                   |
| 4,5,6   | 320                  | 0.0012                                   |
| 5,6,7   | 896                  | 0.0034                                   |
| 6,7,8   | 960                  | 0.0037                                   |
| 7,8,1   | 768                  | 0.0029                                   |
| 8,1,2   | 480                  | 0.0018                                   |

Table 3.2.: Maximum Probabilities of Collisions at the Output of S-box Triplets.
Let $Z_{\Delta,j}$ denote the set of all possible 18-bit S-box triplet inputs $z_i$, which cause collisions at the output of S-boxes $j, j + 1$ and $j + 2$ for a particular differential $\Delta$. However, depending on the secret 18-bit key $k$ not all S-box triplet inputs $z_i$ can occur, because $k \oplus z_i$ must be a valid 18-bit input which can propagate through the expansion permutation of the DES $f$-function. The term $k \oplus z_i$ must satisfy the following conditions.

$$
(k \oplus z_i)[4] = (k \oplus z_i)[6] \quad (3.11)
$$

$$
(k \oplus z_i[5]) = (k \oplus z_i)[7] \quad (3.12)
$$

$$
(k \oplus z_i[10]) = (k \oplus z_i)[12] \quad (3.13)
$$

$$
(k \oplus z_i[11]) = (k \oplus z_i)[13] \quad (3.14)
$$

For a given $z_i$ eight bits $k[4], k[5], k[6], k[7]$ and $k[10], k[11], k[12], k[13]$ of the key $k$ determine whether $k \oplus z_i$ results in a valid input value to the $f$-function. With regard to these eight key bits any 18-bit key $k$ can be partitioned into one of $2^8$ possible key subsets $K_l$ of order $|K_l| = 2^10$ with $l \in \{0, \ldots, 2^8 - 1\}$. In general, we choose only use those differentials$^8$ $\Delta$, for which there exist valid inputs $x_i = E^{-1}(k \oplus z_i)$ for all keys $k \in \{0, \ldots, 2^{18} - 1\}$ where $E^{-1}$ denotes the inverse expansion permutation of the DES $f$-function.

Let us assume an adversary detects a collision at the output of S-box triplet $j$ for some plaintext pair $(x, x \oplus E^{-1}(\Delta))$ with $x \in \{0, \ldots, 2^{14} - 1\}$. For a fixed $x$, let $K$ denote the set of all possible key candidates $k_i$.

$$
K = \{E(x) \oplus z_i = k_i\} \quad z_i \in Z_{\Delta,j} \quad (3.15)
$$

Therefore, the number of key candidates $k_i$ is equal to the number of possible S-box triplet inputs $z_i$.

$$
|K| = |Z_{\Delta,j}| \quad (3.16)
$$

Let $Z_{\Delta,j,K_l}$ denote the subset of valid S-box triplet inputs $z_i$, which can cause a collision for a given key $k \in K_l$.

$$
Z_{\Delta,j,K_l} \subseteq Z_{\Delta,j} \quad l \in \{0, \ldots, 2^8 - 1\} \quad (3.17)
$$

For a given key $k \in K_j$ and a random $x$ the probability of a collision is

$$
p(f(x) = f(x \oplus E^{-1}(\Delta)|k \in K_l) = \frac{|Z_{\Delta,j,K_l}|}{2^{14}} \quad (3.18)
$$

The probability that it takes $M$ trials to find an input $x$ which results in a collision $f(x) = f(x \oplus E^{-1}(\Delta))$ is

$$
p(f(x_M) = f(x_M \oplus E^{-1}(\Delta)|k \in K_l) = \left(1 - \frac{|Z_{\Delta,j,K_l}|}{2^{14}}\right)^{M-1} \cdot \frac{|Z_{\Delta,j,K_l}|}{2^{14}} \quad (3.19)
$$

$^8$Please see [Sch02] for a list of those subkeys, which thwart internal collision attacks for certain input differentials.
The expected number of trials that are required to find an input $x$ which results in a collision for a given key $k \in K_i$ can be found by substitution of the generating function $\sum_{M=1}^\infty M \cdot q^{M-1} = (1 - q)^{-2}$ (see [Funb]).

$$
\mathcal{M}_k = \sum_{M=1}^\infty M \cdot p(f(x_M) = f(x_M \oplus E^{-1}(\Delta)) | k \in K_i)
$$

$$
= \frac{|Z_{\Delta,j,K_i}|}{2^{14}} \cdot \sum_{M=1}^\infty M \cdot \left(1 - \frac{|Z_{\Delta,j,K_i}|}{2^{14}}\right)^{M-1}
$$

$$
= \frac{|Z_{\Delta,j,K_i}|}{2^{14}} \cdot \frac{\left(\frac{|Z_{\Delta,j,K_i}|}{2^{14}}\right)^{-2}}{\frac{|Z_{\Delta,j,K_i}|}{2^{14}}} = \frac{2^{14}}{|Z_{\Delta,j,K_i}|} \quad (3.20)
$$

The total probability of a collision for an unknown key $k \in \{0, ..., 2^{18} - 1\}$ is

$$
p(f(x) = f(x \oplus E^{-1}(\Delta))) = \sum_{l=0}^{255} p(f(x) = f(x \oplus E^{-1}(\Delta)) | k \in K_i) \cdot p(k \in K_i)
$$

$$
= 2^{-22} \cdot \sum_{l=0}^{255} |Z_{\Delta,j,K_i}| \quad (3.21)
$$

The expected number of trials until a collision occurs for any key $k \in \{0, ..., 2^{18} - 1\}$ is

$$
\overline{M} = \frac{1}{256} \cdot \sum_{l=0}^{255} \mathcal{M}_k = 2^6 \cdot \sum_{l=0}^{255} \frac{1}{|Z_{\Delta,j,K_i}|} \quad (3.22)
$$

Since every collision trial requires two inputs, i.e. $x$ and $x \oplus E^{-1}(\Delta)$, the expected number of encryptions, i.e. complexity $C$, to find a collision for an unknown key $k$ is

$$
C = 2 \cdot \overline{M} = 2^7 \cdot \sum_{l=0}^{255} \frac{1}{|Z_{\Delta,j,K_i}|} \quad (3.23)
$$

### 3.2.3. Optimization of the Collision Attack against DES

In order to decrease the number of encryptions until a collision occurs the attack can be extended to $v$ differentials $\Delta_1, \ldots, \Delta_v$ yielding a set of $2^v$ possible encryptions $f(x)$, $f(x \oplus E^{-1}(\Delta_1))$, $f(x \oplus E^{-1}(\Delta_2))$, $f(x \oplus E^{-1}(\Delta_2 \oplus \Delta_1))$, …, $f(x \oplus E^{-1}(\Delta_v \oplus \ldots \oplus \Delta_1))$ for a given $x$. In this case all possible encryption pairs are tested for collisions which increases the likelihood of a collision due to the birthday paradox. A collision $f(x') = f(x'')$ can only occur, if $E(x' \oplus x'')$ equals a known differential $\Delta_i$, with $i \in \{1, \ldots, v\}$. In Table 3.3 the costs of the attacks using a single differential $\Delta$ and $v$ differentials $\Delta_1, \ldots, \Delta_v$ are compared. For example, using a single $\Delta$ the random generation of $m = 64$ inputs $x_j$ will
result in $C = 128$ encryptions, but will only yield 64 collision trials $f(x) = f(x \oplus E^{-1}(\Delta))$. Using $v = 4$ differentials $\Delta_1, \ldots, \Delta_4$ the random generation of $m = 8$ inputs $x$ will also result in $C = 8 \cdot 2^4 = 128$ encryptions, but will yield $8 \cdot 4 \cdot 2^3 = 256$ collision tests. In this example, the same number of encryptions results in a fourfold number of collision trials and, thus, in a greater overall collision probability.

As another example, Figure 3.5 shows a set of $2^v = 2^3 = 8$ encryptions for $v = 3$ differentials $\Delta_1$, $\Delta_2$ and $\Delta_3$. In this case $v \cdot 2^{v-1} = 3 \cdot 2^2 = 12$ possible collisions $f(x)$, $f(x \oplus \Delta_1)$, $f(x \oplus \Delta_2)$, $f(x \oplus \Delta_3 \oplus \Delta_1)$, $f(x \oplus \Delta_3)$, $f(x \oplus \Delta_3 \oplus \Delta_2)$, $f(x \oplus \Delta_1)$, $f(x \oplus \Delta_3 \oplus \Delta_2)$, $f(x \oplus \Delta_3 \oplus \Delta_2 \oplus \Delta_1)$

![Diagram showing possible collision tests for $n = 3$ differentials](image)

In this example the probability that at least one collision occurs (under the condition that all collision trials are statistically independent, see [Mihi00]) is

$$p((A1 \cup A2 \cup A3 \cup A4) \cup (B1 \cup B2 \cup B3 \cup B4) \cup (C1 \cup C2 \cup C3 \cup C4)) = 1 - (1 - p_A)^4 \cdot (1 - p_B)^4 \cdot (1 - p_C)^4$$

$$\approx 4 \cdot p_A + 4 \cdot p_B + 4 \cdot p_C \quad \text{if} \quad p_A, p_B, p_C << 1$$
3.2 Internal Collisions in DES

In general, if \( v \) differentials are being used in an attack, and, there exist no statistical dependencies among the \( v \cdot 2^{v-1} \) collision trials, and, the probability of a collision in any of these trials is reasonable small, then the probability that at least one collision occurs is

\[
p(\text{collision}) = 1 - \prod_{i=1}^{v} (1 - p_i)^{2^{v-1}} \approx 2^{v-1} \cdot \sum_{i=1}^{v} p_i
\]  

(3.24)

with \( p_i = p(f(x) = f(x \oplus E^{-1}(\Delta_i))) \). As mentioned above, we presume that collision tests are statistically independent, i.e. the occurrence of a particular collision does not condition any other collision within a set of trials. Surprisingly, analysis of the collision sets \( Z_{\Delta,j} \) revealed that statistical dependencies among collision trials do exist for certain differentials [Sch02]. In general, statistical dependent collision trials are not desired, because they decrease the overall probability of a collision within a set of trials.

We also discovered that there exist many linear combinations among the differentials for all eight S-box triplets. In an attack based on multiple differentials \( \Delta_1, \ldots, \Delta_v \) linear combinations of these can thus lead to additional valid differentials \( \Delta_i^* \) and thus additional collision trials without the necessity to increase the number of encryptions.

\[
\Delta_i^* = a_1 \cdot \Delta_1 \oplus \ldots \oplus a_v \cdot \Delta_v, a_i \in \{0, 1\} \quad \Delta_i \neq \Delta_1 \neq \ldots \neq \Delta_v
\]  

(3.25)

**Example:** An adversary tries to cause a collision in S-boxes 2,3,4 using \( v = 5 \) input differentials \( \Delta_3, \Delta_{13}, \Delta_{15}, \Delta_{16} \) and \( \Delta_{21} \). The values of these 18-bit input differentials and the corresponding S-box triplet input values can be looked up in given tables\(^9\).

\[
\begin{align*}
\Delta_3 &= 000011110110101000_2 \\
\Delta_{13} &= 00011111010101000_2 \\
\Delta_{15} &= 00011111110101000_2 \\
\Delta_{16} &= 00011111110101100_2 \\
\Delta_{21} &= 00101111010101000_2
\end{align*}
\]

Analysis of these tables for the S-box triplet 2,3,4 reveals that the following linear combinations exist.

\[
\begin{align*}
\Delta_1 &= \Delta_3 \oplus \Delta_{13} \oplus \Delta_{15} \\
\Delta_2 &= \Delta_3 \oplus \Delta_{13} \oplus \Delta_{16} \\
\Delta_4 &= \Delta_3 \oplus \Delta_{15} \oplus \Delta_{16} \\
\Delta_{14} &= \Delta_{13} \oplus \Delta_{15} \oplus \Delta_{16} \\
\Delta_{22} &= \Delta_{15} \oplus \Delta_{16} \oplus \Delta_{21} \\
\Delta_{23} &= \Delta_{13} \oplus \Delta_{15} \oplus \Delta_{21} \\
\Delta_{24} &= \Delta_{13} \oplus \Delta_{16} \oplus \Delta_{21}
\end{align*}
\]

\(^9\)See the appendix of [Sch02] for a further description.
These seven linear combinations allow the adversary to check $7 \cdot 2^{v-1} = 112$ additional collision tests in each set of $2^v = 32$ encryptions. The total number of collision trials for a set of 32 encryptions is thus $(v + 7) \cdot 2^{v-1} = 192$ and, as a result, the overall probability of a collision is significantly increased.

After a first collision has been observed further collisions will provide additional key candidate sets $K_i$. For example, after $c$ collisions have been observed the intersection

$$K_{int} = K_1 \cap K_2 \cap \ldots \cap K_c$$

(3.26)

Additional collisions can be easily found in an S-box triplet by fixing the 12-bit input of two S-boxes and only varying the input of the third S-box. Due to the expansion permutation in the $f$-function not all input bits of the third S-box can be varied, though. Only bits $2-5$ of the left S-box, bits $2$ and $3$ of the middle S-box and bits $0-3$ of the right S-box of the triplet can be varied without altering the inputs of the remaining two S-boxes. This is shown in Figure 3.6. With regard to each of the three S-boxes of an S-box triplet a maximum$^{10}$ of $15+3+15 = 33$ additional S-box triplet inputs can be tested for collisions.

![Figure 3.6: Additional collisions in one of the three S-boxes while preserving the state of the two remaining S-boxes.](image)

*Example:* An adversary tries to cause collisions in S-boxes 1, 2, 3 using the input differential $\Delta_3 = 000011110010101100_2$. A first collision $f(x) = f(x \oplus E^{-1}(\Delta_3))$ yields $|Z_{\Delta_3}| = 1120$ possible key candidates. Analysis of the collision set $Z_{\Delta_3}$ reveals that there exist 18 out of 33 differentials $\epsilon_i$, which, as aforementioned, only change the input of a single S-box of the triplet. As a result, the adversary is able to find several additional collisions $f(x \oplus E^{-1}(\epsilon_i)) = f(x \oplus E^{-1}(\epsilon_i \oplus \Delta_3))$, which drastically decrease the number of key candidates from 1120 down to 16.

In a computer simulation, we searched for the optimum combination of differentials $\Delta_i$ for all eight S-box triplets in order to minimize the number of required encryptions until a first collision occurs. The results of this exhaustive search are listed in Table 3.4, where $\overline{C}$ denotes the average$^{11}$ number of encryptions until a collision occurs. $\overline{K}$ denotes the average number of key candidates corresponding to 18 key bits found after applying the key reduction method stated above. As the best result, we were able to cause a collision

---

$^{10}$As a matter of fact, this maximum does never occur due to the S-box design criteria of DES.

$^{11}$Averaged over $10,000 = 10^4$ random keys
in S-box triplet 2,3,4 with an average of 140 encryptions. Using the key reduction method we were able to delimit 18 key-bits to an average of $220 \approx 2^{7.8}$ key bits, i.e., 10.2 key bits were broken. Moreover, we were able to cause collisions in S-box triple 7,8,1 with an average of 165 encryptions yielding an average of $19 \approx 13.8$ key bits.

<table>
<thead>
<tr>
<th>S-boxes</th>
<th>$v$</th>
<th>$\Delta_i$</th>
<th>$C$</th>
<th>$K$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,3</td>
<td>3</td>
<td>$\Delta_3, \Delta_{15}, \Delta_{18}$</td>
<td>227</td>
<td>20</td>
</tr>
<tr>
<td>2,3,4</td>
<td>5</td>
<td>$\Delta_3, \Delta_{13}, \Delta_{15}, \Delta_{16}, \Delta_{21}$</td>
<td>140</td>
<td>220</td>
</tr>
<tr>
<td>3,4,5</td>
<td>3</td>
<td>$\Delta_3, \Delta_{10}, \Delta_{12}$</td>
<td>190</td>
<td>110</td>
</tr>
<tr>
<td>4,5,6</td>
<td>3</td>
<td>$\Delta_2, \Delta_{10}, \Delta_{11}$</td>
<td>600</td>
<td>71</td>
</tr>
<tr>
<td>5,6,7</td>
<td>5</td>
<td>$\Delta_2, \Delta_5, \Delta_8, \Delta_{23}, \Delta_{29}$</td>
<td>200</td>
<td>24</td>
</tr>
<tr>
<td>6,7,8</td>
<td>5</td>
<td>$\Delta_7, \Delta_{10}, \Delta_{19}, \Delta_{20}, \Delta_{22}$</td>
<td>186</td>
<td>52</td>
</tr>
<tr>
<td>7,8,1</td>
<td>5</td>
<td>$\Delta_1, \Delta_2, \Delta_7, \Delta_{17}, \Delta_{19}$</td>
<td>165</td>
<td>19</td>
</tr>
<tr>
<td>8,1,2</td>
<td>4</td>
<td>$\Delta_1, \Delta_2, \Delta_8, \Delta_{38}$</td>
<td>208</td>
<td>158</td>
</tr>
</tbody>
</table>

Table 3.4.: Best results of internal collision attacks against all eight possible S-box triplets. Each attack used $v$ 18-bit input differentials $\Delta_i$ and on average revealed $K$ key candidates after $C$ encryptions. The results were averaged over $10^4$ random keys.

3.2.4. Attacking a Software Implementation of DES

In Figure 3.7, the propagation path of a collision occurring in the $f$-function of round $n$ is shown. In this case the $f$-function in round $(n+1)$ processes the same input data\(^1\). In general, an adversary tries to cause collisions in the first round ($n = 1$) or last round ($n = 16$), because he has full access to the input of the $f$-function in a chosen plaintext/ciphertext attack.

In [SWP03], it was shown that collisions at the output of the $f$-function of DES can be detected with side channel analysis. Let us assume that an adversary randomly varies exactly those 14 input bits of the $f$-function in round one, which enter a chosen S-box triplet. All 50 remaining bits of the plaintext are fixed. The $f$-function expands the 14-bit input $x$ to the 18-bit input $E(x)$, which is x-ored with 18 bits of the round key $k$. The result $z = E(x) \oplus k$ enters the chosen S-box triplet. We assume that the adversary uses power analysis to monitor the power consumption of the cryptographic device during round two. Next, he/she encrypts the input $(E(x) \oplus \Delta)$ and again monitors the power consumption during round two. A high correlation of the two observed power traces reveals that the same data was processed, i.e., a collision occurred. As mentioned

\(^1\)Collisions in succeeding rounds can also occur, but with vanishing probability. Differential cryptanalysis of DES proposed by Biham and Shamir is based on collisions in every other round, so-called two round characteristics [BS90].
Figure 3.7.: Propagation path of an internal collision in the $f$-function of DES.
in Section 3.2.2, analysis of the differential characteristics of the S-boxes then reveals possible key candidates $k_i = z_i \oplus E(x)$.

In order to verify the internal collision attack, we attacked an assembly implementation of DES running on a 8051 microcontroller. The measurement setup used in the attack is shown in Figure 3.8. In this setup a PC sends chosen plaintexts to the microcontroller and, thus, triggers new encryptions. In order to measure the power consumption of the microcontroller a small shunt resistance ($R_s = 10\Omega$) was put in series between the ground pad of the microcontroller and the ground of the power supply. We also replaced the original voltage source of the microcontroller with a low-noise voltage source to minimize noise superimposed by the source. The digital oscilloscope HP1662AS was used to sample the voltage over the shunt resistance at 1 GHz. Collisions were caused in the first round of DES. Power traces of round two were transferred to the PC using the GPIB interface. The PC was used to cross-correlate power traces of different encryptions in order to detect collisions. In our experiments we discovered that a correlation coefficient approximately greater than 0.95 typically indicated a collision. If no collision occurred, the correlation coefficient was always well below 0.95, typically less than 0.80. In general, uncorrelated noise such as noise caused by the voltage source, quantization noise of the oscilloscope or intrinsic noise within the microcontroller can be decreased by averaging power traces of equal encryptions\textsuperscript{13}. In our experiments we found out that averaging as few as $N = 10$ power traces was already sufficient to achieve the correlation results stated above. For example, in Figures 3.9 and 3.10 the averaged power traces of two different plaintext encryptions $x$ and $(x \oplus \Delta)$ during the S-box look-up in round two are shown. The power traces 3.9 and 3.10 clearly differ in peaks in the time frame between 0 and 1000 ns, which indicates that no collision occurred.

\textsuperscript{13}We assume that no countermeasures such as random dummy cycles are present.
Figure 3.9.: Power consumption of the microcontroller encrypting $x$ during the S-box look-up in round two.

Figure 3.10.: Power consumption of the microcontroller encrypting $(x \oplus \Delta)$ during the S-box look-up in round two. The power trace differs from the trace shown above in the time frame between 0 and 1000 ns, which indicates that no collision occurred.
3.2.5. A Cryptographically Strengthened S-box Resistant to Collision Attacks

In [Pos05, LPPS06], we propose a lightweight variant of DES which substitutes the eight original S-boxes of DES with a single S-box repeated eight times. The S-box we selected is more resistant against linear and differential cryptanalysis and, also, resistant against internal collision attacks. One step to improve the resistance of DES against linear cryptanalysis was already proposed by Coppersmith in [Cop94]. He defined a stronger criterion (S-2') regarding the linearity of the S-boxes as follows:

(S-2') No combination of output bits of an S-box should be too close to a linear function of the input bits. (That is, if we select any subset of the four output bit positions and any subset of the six input bits, the fraction of inputs for which this input equals the XOR of these input bits should not be close to 0 or 1, but rather should be near $\frac{1}{2}$.)

All possible linear combinations of input bits $x$ and output bits $S(x)$ can be represented by the scalar products $\langle a, x \rangle$ and $\langle b, S(x) \rangle$, with $a, x \in GF(2)^6$ and $b, S(x) \in GF(2)^4$, respectively. Let $S_b = \langle b, S(x) \rangle$ denote a combination of output bits, that is determined by $b$. Then, the Walsh coefficient $S_b^w(a)$ is a measure for the linear approximation of the output combination $S_b$ by an input combination $a$ [Dob01].

$$S_b^w(a) = \sum_{x \in GF(2)^6} (-1)^{(b,S(x))+(a,x)}$$  \hspace{1cm} (3.27)

The probability $p$ that a linear combination of output bits $S_b$ is equal to a linear combination of input bits can be written as

$$p = \frac{\# \{x | S_b(x) = \langle a, x \rangle \}}{2^6}$$  \hspace{1cm} (3.28)

Combining equations 3.27 and 3.28 leads to

$$p = \frac{S_b^w(a)}{2^7} + \frac{1}{2}$$  \hspace{1cm} (3.29)

The linear probability bias $\varepsilon$ is a correlation measure for this deviation from probability $\frac{1}{2}$ for which it is entirely uncorrelated.

$$\varepsilon = \left| p - \frac{1}{2} \right| = \left| \frac{S_b^w(a)}{2^7} \right|$$  \hspace{1cm} (3.30)

Let us denote the maximum value derived from the Walsh transformation by $S_{2 max}$.

$$\varepsilon = \left| \frac{S_{2 max}(a)}{2^7} \right|$$  \hspace{1cm} (3.31)
The maximum deviation $\varepsilon$ plays an important role in linear cryptanalysis [Mat94]. In general, a smaller linear bias $\varepsilon$ (and hence a smaller value of $S^{2\text{max}}$) results in an S-box, which is more resistant against linear cryptanalysis. The biases of the original eight DES S-boxes are shown in Table 3.2.5. As one can see, no S-box exhibits a bias $S^{2\text{max}}$ smaller than 28. Among all S-boxes, S-box 5 features the greatest maximum bias $S^{2\text{max}} = 40$, which is exploited in Matsui’s linear attack [Mat94].

<table>
<thead>
<tr>
<th>Combination of output bits $b$</th>
<th>Maximum bias $S^{2\text{max}}$ for S-box</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S1</td>
</tr>
<tr>
<td>$x_0$</td>
<td>28</td>
</tr>
<tr>
<td>$x_1$</td>
<td>24</td>
</tr>
<tr>
<td>$x_1 \oplus x_0$</td>
<td>16</td>
</tr>
<tr>
<td>$x_2$</td>
<td>20</td>
</tr>
<tr>
<td>$x_2 \oplus x_0$</td>
<td>20</td>
</tr>
<tr>
<td>$x_2 \oplus x_1$</td>
<td>24</td>
</tr>
<tr>
<td>$x_2 \oplus x_1 \oplus x_0$</td>
<td>24</td>
</tr>
<tr>
<td>$x_3$</td>
<td>28</td>
</tr>
<tr>
<td>$x_3 \oplus x_0$</td>
<td>16</td>
</tr>
<tr>
<td>$x_3 \oplus x_1$</td>
<td>24</td>
</tr>
<tr>
<td>$x_3 \oplus x_1 \oplus x_0$</td>
<td>24</td>
</tr>
<tr>
<td>$x_3 \oplus x_2$</td>
<td>24</td>
</tr>
<tr>
<td>$x_3 \oplus x_2 \oplus x_0$</td>
<td>20</td>
</tr>
<tr>
<td>$x_3 \oplus x_2 \oplus x_1$</td>
<td>24</td>
</tr>
<tr>
<td>$x_3 \oplus x_2 \oplus x_1 \oplus x_0$</td>
<td>36</td>
</tr>
<tr>
<td><strong>maximum</strong></td>
<td>36</td>
</tr>
</tbody>
</table>

Table 3.5.: Maximum biases $S^{2\text{max}}$ of the original eight DES S-boxes.

The stronger criterion (S-2") still does not include a maximum threshold, which defines how close to $p = \frac{1}{2}$ the correlation of linear combinations of input and output bits should be. We define a stronger criterion (S-2") by setting the upper bound of $S^{2\text{max}}$ to 28:

(S-2") No combination of output bits of an S-box should have a linear probability bias greater than $\frac{28}{128} (\varepsilon \leq \frac{7}{32})$.

As proposed by Biham and Shamir in [BS90], collisions at the output of S-box triplets are exploited in differential cryptanalysis of DES as two-round characteristics. Better than minimizing the probability of collisions in three or more adjacent S-boxes by modifying the (S-7) criterion\(^{14}\), is to eliminate them, entirely. Consider an input difference $\Delta I_{i,j}$ of

\(^{14}\)As a reminder: the (S-7) criterion states that the maximum number of inputs of an S-box which result in a collision at the output for a fixed input differential is limited to 16, see [Cop94].
an S-box \( i \) which results in an output difference \( \Delta O_{i,j} = 0 \):

\[
\Delta I_{i,j} = abcdef,
\]

where \( a, b, c, d, e, f \) are arbitrary bits. If S-box \( i \) is the rightmost active S-box of an S-box tuple and there are seven or less active S-boxes, then input bits \( e \) and \( f \) have to be 0 in order not to alter the input of the inactive S-box to the right.

\[
\Delta I_{i,j} = abcd00
\]

Design criterion (S-4) states, that there are no collisions in one row of an S-box, hence \( a \) has to be 1.

\[
\Delta I_{i,j} = 1bcd00
\]

This is always the input difference of the rightmost active S-box for any number of adjacent S-boxes except for eight adjacent active S-boxes. If there are no collisions with such kind of input differences, differential attacks using differentials such as the ones presented by Biham and Shamir in [BS90], will not be successful. Hence, we replace (S-6) and (S-8') by the new design criterion (S-6'):

(\( S-6' \)) If two inputs to an S-box differ in their first bit and are identical in their last two bits, the two outputs must not be the same.

(If \( \Delta I_{i,j} = 1xyz00 \), where \( x, y \) and \( z \) are arbitrary bits, then \( \Delta O_{i,j} \neq 0 \).)

Note that the pattern \( \Delta I_{i,j} = 11xyz00 \) used for the rightmost input differential in criterion (S-8) is a special case of the input difference \( \Delta I_{i,j} = 1xyz00 \) used in (S-6'). Hence, collisions in up to seven adjacent S-boxes will not be possible. We randomly generated S-boxes, which fulfill the original DES criteria (S-1), (S-3), (S-4), (S-5), (S-7), and the newly defined criteria (S-2’’) and (S-6’). We chose an S-box which features a maximum linear bias of \( S_{2_{\text{max}}} = 28 \) (S-2”') and a maximum occurrence of 7 S-box input pairs for a fixed input and output difference (S-7). Table 3.2.5 shows the best S-box we found in 1000 S-boxes, which satisfies all new criteria. During the search, more than 200 million S-boxes were discarded.

<table>
<thead>
<tr>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>13</td>
</tr>
</tbody>
</table>

Table 3.6.: Strengthened DES S-box resistant against internal collision attacks.
3.3. Internal Collisions in AES

As discussed in Section 3.1, collisions can only occur at the output of functions, which are non-injective. However, partial collisions may also occur at the output of functions, which are not non-injective. In the case of AES we will show that key-dependent collisions can occur in one of the four output bytes of the MixColumn function [SLFP04]. In general, for each observed collision reveals 8 bits of the secret key.

In Section 3.3.1, we briefly review the MixColumn function of AES and show that key-dependent collisions can occur in one of its output bytes. In Section 3.3.2, we propose an optimized variant of the attack. We show that precomputed tables of a total size of 540 MB can be used to derive an average of 32 key bits with only 31 encryptions. Furthermore, if the attack is applied in parallel to all four columns, 128 key bits can be extracted with an average of only 40 encryptions. Finally, in Section 3.3.4, we give details about our simulated and real-world attacks against a 8051-based microcontroller running AES in assembly.

3.3.1. Collisions in the MixColumn Transformation

The MixColumn transformation is linear and bijective and its main purpose in AES is diffusion. It maps a four-byte column vector to a four-byte column vector. Throughout this paper we follow the notation used in [DR02]. The mathematical background of the MixColumn transformation is as follows: 4-byte columns are considered as polynomials over $GF(2^8)$ modulo $m(x) = x^8 + x^4 + x^3 + x + 1$. The input polynomial is multiplied with the fixed polynomial

$$c(y) = 03 \cdot y^3 + 01 \cdot y^2 + 01 \cdot y + 02$$

where 01, 02 and 03 refer to the $GF(2^8)$ elements 1, $x$ and $x + 1$, respectively. If we refer to the input column as $a(y)$ and to the output column as $b(y)$, the MixColumn transformation can be stated as

$$b(y) = a(y) \times c(y) \mod y^4 + 1$$

This specific multiplication with the fixed polynomial $c(y)$ can also be written as a matrix multiplication

$$\begin{pmatrix} b_{00} \\ b_{10} \\ b_{20} \\ b_{30} \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \end{pmatrix}$$

If we look at the first output byte $b_{00}$, it is given by\textsuperscript{15}

$$b_{00} = 02 \cdot a_{00} + 03 \cdot a_{10} + 01 \cdot a_{20} + 01 \cdot a_{30}$$

\textsuperscript{15}The symbol $+$ denotes an addition modulo 2, i.e. the binary $\lor$-addition.
If we focus on the first round, we can substitute $a_{00}, a_{10}, a_{20}$ and $a_{30}$ with $S(p_{00} + k_{00}), S(p_{11} + k_{11}), S(p_{22} + k_{22})$ and $S(p_{33} + k_{33})^{16}$. The output byte $b_{00}$ can then be written as

$$b_{00} = 02 \cdot S(p_{00} + k_{00}) + 03 \cdot S(p_{11} + k_{11}) + 01 \cdot S(p_{22} + k_{22}) + 01 \cdot S(p_{33} + k_{33})$$

The main idea of this attack is to find two different plaintext pairs with the same output byte $b_{00}$. We are only considering plaintexts with $p_{22} = p_{33}$ while all remaining plaintext bytes, such as $p_{00}$ and $p_{11}$, are assumed to be constant for right now. If two plaintexts with $p_{22} = p_{33} = \delta$ and $p'_{22} = p'_{33} = \epsilon \neq \delta$ result in an equal output byte $b_{00}$, the following equation is satisfied:

$$S(\delta + k_{22}) + S(\delta + k_{33}) = S(\epsilon + k_{22}) + S(\epsilon + k_{33})$$

Suppose that an adversary has the necessary experience and measurement instrumentation to detect this collision in $b_{00}$ (or any other output byte of the mix column transformation) with side channel analysis. First, he sets the two plaintext bytes $p_{22}$ and $p_{33}$ to a random value $\delta = p_{22} = p_{33}$. As next, he encrypts the corresponding plaintext, measures the power trace and stores it on his computer. He then keeps generating new random values $\epsilon = p'_{22} = p'_{33}$ unequal to previously generated values of $\delta, \epsilon$, and so on. He encrypts each new plaintext, measures and stores the corresponding power trace and cross-correlates it with all previously stored power traces until he detects a collision in output byte $b_{00}$. Once a collision has been found the task is to deduce information about $k_{22}$ and $k_{33}$.

### 3.3.2 An Analysis of the Collision Function

To simplify the notation, we denote $k_{00}$ (or $k_{11}, k_{22}, k_{33}$) simply by $k_0$ (or $k_1, k_2, k_3$) and output byte $b_{00}$ by $b_0$. As described above, we are interested in values $(\delta, \epsilon)$, such that for an unknown key the following equation is satisfied:

$$S(k_2 + \delta) + S(k_3 + \delta) + S(k_2 + \epsilon) + S(k_3 + \epsilon) = 0 \quad (3.32)$$

Set

$$\mathcal{L}_{(a,b)} = \{(x, y) \in \mathbb{F}_2^{28} \times \mathbb{F}_2^{28} \mid S(a + x) + S(b + x) + S(a + y) + S(b + y) = 0\} \quad (3.33)$$

The interpretation of this set is twofold. Given a key pair $(k_2, k_3)$, the set $\mathcal{L}_{(k_2, k_3)}$ is the set of all pairs $(\delta, \epsilon)$, which will lead to a collision in $b_0$. On the other hand, due

---

16These are the diagonal elements of the plaintext and initial round key matrix due to the prior shift row transformation.
to symmetry, the set $L_{(\delta, \epsilon)}$ contains all possible key pairs, for which $(\delta, \epsilon)$ will lead to a collision in byte $b_0$.

Note that if we measure a collision for $\delta$ and $\epsilon$, the key $(k_2, k_3)$ cannot be uniquely determined. This is due to the following properties of the set $L_{(a, b)}$:

$$\forall x \in \mathbb{F}_2^8 \quad (x, x) \in L_{(a, b)}$$
$$\quad (x, y) \in L_{(a, b)} \Rightarrow (y, x) \in L_{(a, b)}$$
$$\quad (x, y), (y, c) \in L_{(a, b)} \Rightarrow (x, c) \in L_{(a, b)}$$
$$\quad (x, y) \in L_{(a, b)} \Rightarrow (x, y + a + b) \in L_{(a, b)}$$

Equations (1) to (3) establish an equivalence relation on $\mathbb{F}_2^8$.

More explicitly, if $(k_2, k_3) \in L_{(\delta, \epsilon)}$, it follows that

$$(k_2 + \delta + \epsilon, k_3) \in L_{(\delta, \epsilon)}$$
$$(k_2, k_3 + \delta + \epsilon) \in L_{(\delta, \epsilon)}$$
$$(k_2 + \delta + \epsilon, k_3 + \delta + \epsilon) \in L_{(\delta, \epsilon)}$$
$$(k_3, k_2) \in L_{(\delta, \epsilon)}$$
$$(k_3 + \delta + \epsilon, k_2) \in L_{(\delta, \epsilon)}$$
$$(k_3, k_2 + \delta + \epsilon) \in L_{(\delta, \epsilon)}$$
$$(k_3 + \delta + \epsilon, k_2 + \delta + \epsilon) \in L_{(\delta, \epsilon)}$$

and thus, we cannot hope to determine $k_2$ and $k_3$ completely given one collision.

Let $(\delta, \epsilon) \in L_{(k_2, k_3)}$ where we always assume that $\epsilon \neq \delta$. We have to discuss several cases:

**case 1:** If $k_2 = k_3$ then $L_{(k_2, k_3)} = \mathbb{F}_2^8 \times \mathbb{F}_2^8$, every choice of $(\delta, \epsilon)$, i.e., every measurement will lead to a collision.

**case 2:** If $k_2 \neq k_3$ and if we furthermore assume that $\delta, \epsilon \notin \{k_2, k_3\}$ we obtain

$$0 = S(k_2 + \delta) + S(k_3 + \delta) + S(k_2 + \epsilon) + S(k_3 + \epsilon).$$

By expressing $S(x)$ as $L(x^{-1})$ and applying $L^{-1}$ (where $L$ is the affine transformation of the S-box) we conclude

$$0 = \frac{1}{k_2 + \delta} + \frac{1}{k_3 + \delta} + \frac{1}{k_2 + \epsilon} + \frac{1}{k_3 + \epsilon}$$

which finally yields

$$k_2 + k_3 = \delta + \epsilon.$$  

**case 3:** If $k_2 = \delta$ and $k_3 = \epsilon$ or $k_3 = \delta$ and $k_2 = \epsilon$, we also conclude that $k_2 + k_3 = \delta + \epsilon$.
3.3 Internal Collisions in AES

**case 4:** This case occurs if either $k_2 \in \{\delta, \epsilon\}$ or $k_3 \in \{\delta, \epsilon\}$. If $k_2 \in \{\delta, \epsilon\}$, we compute

$$p(k_3) = \frac{k_3^2}{(\delta + \epsilon)^2} + \frac{k_3}{\delta + \epsilon} + \frac{\delta \epsilon}{(\delta + \epsilon)^2} + 1 = 0 \quad (3.41)$$

This can be further simplified to

$$p(k_3) = \left(\frac{k_3 + \delta}{\delta + \epsilon}\right)^2 + \frac{k_3 + \delta}{\delta + \epsilon} + 1 = 0 \quad (3.42)$$

which shows that

$$\alpha = \frac{k_3 + \delta}{\delta + \epsilon} \in \mathbb{F}_4 \setminus \{1\}$$

An analysis of the case $k_3 \in \{\delta, \epsilon\}$ yields a similar result. Combining both cases, we deduce the following possibilities for $(k_2, k_3)$

\[
\begin{align*}
\text{if } k_2 &= \delta \quad \text{and } k_3 = \alpha(\delta + \epsilon) + \delta \\
\text{if } k_2 &= \epsilon \quad \text{and } k_3 = \alpha(\delta + \epsilon) + \delta \\
\text{if } k_2 &= \delta \quad \text{and } k_3 = \alpha(\delta + \epsilon) + \epsilon \\
\text{if } k_2 &= \epsilon \quad \text{and } k_3 = \alpha(\delta + \epsilon) + \epsilon \\
\text{if } k_3 &= \delta \quad \text{and } k_2 = \alpha(\delta + \epsilon) + \delta \\
\text{if } k_3 &= \epsilon \quad \text{and } k_2 = \alpha(\delta + \epsilon) + \delta \\
\text{if } k_3 &= \delta \quad \text{and } k_2 = \alpha(\delta + \epsilon) + \epsilon \\
\text{if } k_3 &= \epsilon \quad \text{and } k_2 = \alpha(\delta + \epsilon) + \epsilon
\end{align*}
\]

where $\alpha \in \mathbb{F}_4 \setminus \{1\}$. In the case of the AES S-box, $\alpha$ can be chosen as $\alpha(x) = BC = x^7 + x^5 + x^4 + x^3 + x^2$. Note that solutions (3.43) to (3.49) correspond exactly to the seven additional possibilities (3.34) to (3.40).

Let us assume we detect a collision for a particular $(\delta, \epsilon) \in \mathcal{L}_{(k_2, k_3)}$. In order to deduce information about $k_2$ and $k_3$ we have to decide which case we deal with. We do not have to distinguish case two and case three, as the information we deduce about $k_2$ and $k_3$ is the same in both cases.

To distinguish case one, two or three from case four we use the following idea. Given a collision $(\delta, \epsilon)$, we construct a new pair $(\delta', \epsilon')$, which will not lead to a collision if and only if $(\delta, \epsilon)$ corresponds to case four. For this we need

**Lemma 3.3.1** Let

$$\mathcal{L}_4 = \{(k_2, \alpha(k_2 + k_3) + k_2), (k_2, \alpha(k_2 + k_3) + k_3), (k_3, \alpha(k_2 + k_3) + k_2), (k_3, \alpha(k_2 + k_3) + k_3), (\alpha(k_2 + k_3) + k_2, k_2), (\alpha(k_2 + k_3) + k_3, k_2), (\alpha(k_2 + k_3) + k_2, k_3), (\alpha(k_2 + k_3) + k_3, k_3)\}.$$
Given an element \((\delta, \epsilon) \in \mathcal{L}_{(k_2, k_3)}\) the pair \((\delta', \epsilon')\) with
\[
\delta' \in \mathbb{F}_{2^8} \setminus \{\delta, \epsilon, \alpha(\delta + \epsilon) + \delta, \alpha(\delta + \epsilon) + \epsilon\}
\]
and
\[
\epsilon' = \delta' + \delta + \epsilon
\]
is in \(\mathcal{L}_{(k_2, k_3)}\) if and only if
\[
k_2 = k_3
\]
or
\[
(\delta, \epsilon) \notin \mathcal{L}_4
\]
i.e. if and only if \((\delta, \epsilon)\) does not correspond to case four.

Proof.

"\(\Leftarrow\)". If \(k_2 = k_3\), the set \(\mathcal{L}_{(k_2, k_3)} = \mathbb{F}_{2^8} \times \mathbb{F}_{2^8}\), so in particular \((\delta', \epsilon') \in \mathcal{L}_{(k_2, k_3)}\). If on the other hand \((\delta, \epsilon) \notin \mathcal{L}_4\), we see that \(\forall \delta' \in \mathbb{F}_{2^8}\), the pair \((\delta', \delta' + \delta + \epsilon) \in \mathcal{L}_{(k_2, k_3)}\).

"\(\Rightarrow\)". Assume \(k_2 \neq k_3\) and \((\delta, \epsilon) \in \mathcal{L}_4\). W.l.o.g. let \(\delta = k_2\) and \(\epsilon = \alpha(k_2 + k_3) + k_2\). If \((\delta', \epsilon') \in \mathcal{L}_{(k_2, k_3)}\) we get
\[
\frac{1}{k_2 + \delta'} + \frac{1}{k_2 + \epsilon' + \alpha(\delta + \epsilon) + \delta + \epsilon} = 0
\]
If we substitute \(k_3 = \alpha(\delta + \epsilon) + \epsilon\) and \(\epsilon' = \delta + \epsilon + \delta'\), we conclude
\[
\frac{1}{\delta + \delta'} + \frac{1}{\delta + \epsilon'} + \frac{1}{\alpha(\delta + \epsilon) + \delta + \epsilon} + \frac{1}{\alpha(\delta + \epsilon) + \epsilon + \epsilon'} = 0
\]
and due to the choice of \(\delta'\) we finally get
\[
\delta + \epsilon = \alpha(\delta + \epsilon)
\]
a contradiction. \(\Box\)

Thus, with the pair \((\delta', \epsilon')\) as constructed in the theorem, we can decide, if \((\delta, \epsilon)\) corresponds to case four or not.

Now we are in a situation where we have to distinguish case one from cases two and three. If \(k_2 \neq k_3\) we see that
\[
D_{k_2, k_3} := \{a + b \mid (a, b) \in \mathcal{L}_{(k_2, k_3)}\}
\]
contains only the values \(k_2 + k_3\) in cases two and three and \(\alpha(k_2 + k_3)\) and \((\alpha + 1)(k_2 + k_3)\) in case four. As a conclusion, we are able to exactly determine in which case we are in order to determine information about \((k_2, k_3)\). In case one if \(k_2 = k_3\) then \(D_{k_2, k_3} = \mathbb{F}_{2^8}\). Thus if we are given a collision \((\delta, \epsilon)\), we choose new values \(\delta''\) such that \(\delta'' + \epsilon \notin \mathcal{L}_4\).
\{\delta + \epsilon, \alpha(\delta + \epsilon), (\alpha + 1)(\delta + \epsilon)\}. As argued above, such a pair \((\delta'', \epsilon)\) will lead to a collision iff \(k_1 = k_2\).

As discussed in Section 3.1, the probability that a collision occurs in one of the output bytes of the MixColumn transformation after \(n\) encryptions is given by

\[
p(n) = 1 - \prod_{i=0}^{n-1} \left( 1 - \frac{i}{256} \right)
\]  

(3.51)

Table 3.7 lists various probabilities of a collision for a different number of encryptions. As a result, due to the birthday paradox on average only 20 encryptions are required in

<table>
<thead>
<tr>
<th>(n)</th>
<th>(p(n))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0.1631</td>
</tr>
<tr>
<td>20</td>
<td>0.5332</td>
</tr>
<tr>
<td>30</td>
<td>0.8294</td>
</tr>
<tr>
<td>40</td>
<td>0.9599</td>
</tr>
<tr>
<td>50</td>
<td>0.9941</td>
</tr>
</tbody>
</table>

Table 3.7: Probability of a collision after \(n\) encryptions

order to get a collision in a single output byte of the MixColumn transformation.

### 3.3.3. Optimization of the AES Collision Attack

In the last section, we described collisions which occur in a single output byte of the mix column transformation. This attack can be optimized by equally varying all four plaintext bytes which enter the mix column transformation while independently observing collisions in one of the four output bytes. Hence, we now try to cause collisions with two pairs of plaintexts of the form \(\delta = p_{00} = p_{10} = p_{20} = p_{30}\) and \(\epsilon = p'_{00} = p'_{10} = p'_{20} = p'_{30}\). We still look for collisions in a single output byte of the mix column transformation, however, now we simultaneously observe all four outputs for collisions.

For example, a collision occurs in the first output byte of the mix column transformation whenever the following equation is fulfilled

\[
C(\delta, \epsilon, k_0, k_1, k_2, k_3) = 02S(k_0 + \delta) + 03S(k_1 + \delta) + S(k_2 + \delta) + S(k_3 + \delta) + 02S(k_0 + \epsilon) + 03S(k_1 + \epsilon) + S(k_2 + \epsilon) + S(k_3 + \epsilon) = 0
\]
We denote, for a known pair \((\delta, \epsilon)\), the set of all solutions by

\[ C_{\delta, \epsilon} := \{(k_0, k_1, k_2, k_3)|C(\delta, \epsilon, k_0, k_1, k_2, k_3) = 0\} \]

Again, suppose that an adversary has the necessary equipment to detect a collision in any of the output bytes \(b_{00}, \ldots, b_{30}\) with side channel analysis. In order to cause collisions in the outputs of the first mix column transformation, he sets the four plaintext bytes \(p_{00}, p_{11}, p_{22}\) and \(p_{33}\) to a random value \(\delta = p_{00} = p_{11} = p_{22} = p_{33}\). As next, he encrypts the corresponding plaintext, measures the power trace and stores it on his computer. He then keeps generating new random values \(\epsilon = p'_{00} = p'_{11} = p'_{22} = p'_{33}\) unequal to previously generated values of \(\delta, \epsilon\), and so on. He encrypts each new plaintext, measures and stores the corresponding power trace and cross-correlates it with all previously stored power traces until he detects a collision in one of the four observed output bytes \(b_{00}, \ldots, b_{30}\). Once a collision has been found the task is to deduce information about \((k_0, k_1, k_2, k_3)\).

This equation can be solved by analysis or by using precomputed look-up tables which contain the solutions \((k_0, k_1, k_2, k_3)\) for particular \((\delta, \epsilon)\). However, an analytic solution is much more complex than the one introduced in the previous section and an analog description is not trivial. An alternative solution to this problem is to create the sets \(C_{\delta, \epsilon}\) for every pair \((\delta, \epsilon)\) by generating all possible values for \((k_0, k_1, k_2, k_3)\) and checking \(C(\delta, \epsilon, k_0, k_1, k_2, k_3) = 0\) for all pairs \((\delta, \epsilon)\).

In our simulations we found that the resulting sets are approximately equal in size and on average contain 16,776,889 \(\approx 2^{24}\) keys, which corresponds to a size of 67 megabytes \(\approx 2^{26}\) bytes per set. Multiplying this with the number of possible \((\delta, \epsilon)\) sets, all sets together would require about 2,000 gigabytes which is only possible with major efforts and distributed storage. Reducing the amount of required disk space and still being able to compute all the necessary information is the purpose of the next section.

Moreover, it must be pointed out that there exist certain keys \((k_0, k_1, k_2, k_3)\) for which no pair \((\delta, \epsilon)\) will result in a collision. To our knowledge, there only exist three classes of keys \((x, x, x, x)\), \((x, x, x, y)\) and \((x, x, y, y)\) which will not result in collisions for any pair \((\delta, \epsilon)\). If the key \((k_0, k_1, k_2, k_3)\) is an element of the key class \((x, x, x, x)\), i.e. all four key bytes are equal, no collisions will occur in any of the four Mix Column output bytes for any pair \((\delta, \epsilon)\) due to the overall required bijectivity of the Mix Column transformation. The probability that this case occurs is \(P = 2^8/2^{32} = 2^{-24}\). If the key \((k_0, k_1, k_2, k_3)\) is an element of the key class \((x, x, y, x)\) or \((x, x, x, y)\) or \((x, x, x, x)\), no collision will occur in the Mix Column output byte \(b_0\). If the key \((k_0, k_1, k_2, k_3)\) is an element of the key class \((x, y, x, x)\) or \((x, y, y, x)\) or \((x, y, x, x)\), no collision will occur in the Mix Column output byte \(b_1\). If the key \((k_0, k_1, k_2, k_3)\) is an element of the key class \((y, x, x, x)\) or \((y, y, x, x)\), no collision will occur in the Mix Column output byte \(b_2\). If the key \((k_0, k_1, k_2, k_3)\) is an element of the key class \((x, x, x, x)\) or \((y, x, x, x)\), no collision will occur in the Mix Column output byte \(b_3\). The probability that any of these cases occurs is \(P = \frac{1}{2^{256}} \cdot \frac{1}{2^{256}} \cdot \frac{1}{2^{256}} \cdot \frac{255}{2^{256}} \approx 2^{-24}\).
Our simulations showed that these are the only exceptional keys which will not result in partial collisions in output bytes $b_0, b_1, b_2$ or $b_3$.

Note that the sets $C_{\delta,\epsilon}$ also contain all the keys which will cause collisions in the output bytes $b_1, b_2$ and $b_3$. Since the entries in the mix column matrix are bytewise rotated to the right in each row, the stored 32-bit keys in the sets $C_{\delta,\epsilon}$ must be cyclically shifted to the right by one, two or three bytes, as well, in order to cause collisions in $b_1$, $b_2$ and $b_3$. Moreover, the amount of space can be further reduced by taking advantage of two different observations. First, we find some dependencies among the elements in a given set $C_{\delta,\epsilon}$ and second we derive a relationship between two sets $C_{\delta,\epsilon}$ and $C_{\delta',\epsilon'}$.

The first approach uses an argument similar to an argument used in Section 3.3.2. If for a fixed pair $(\delta, \epsilon)$ a key $(k_0, k_1, k_2, k_3)$ is in $C_{\delta,\epsilon}$, then the following elements are also in $C_{\delta,\epsilon}$:

\[
(k_0, k_1, k_2, k_3) \in C_{\delta,\epsilon} \implies \\
(k_0, k_1, k_2, k_2) \in C_{\delta,\epsilon} \quad (3.52) \\
(k_0 + \delta + \epsilon, k_1, k_2, k_3) \in C_{\delta,\epsilon} \quad (3.53) \\
(k_0, k_1 + \delta + \epsilon, k_2, k_3) \in C_{\delta,\epsilon} \quad (3.54) \\
(k_0, k_1, k_2 + \delta + \epsilon, k_3) \in C_{\delta,\epsilon} \quad (3.55) \\
(k_0, k_1, k_2, k_3 + \delta + \epsilon) \in C_{\delta,\epsilon} \quad (3.56)
\]

Combining these changes, we find 32 different elements in $C_{\delta,\epsilon}$, given that $k_2 \neq k_3$ and $\delta + \epsilon \neq 0$. The case $\delta + \epsilon = 0$ is à priori excluded. If $k_2 = k_3$, we still find 16 different elements in $C_{\delta,\epsilon}$. For the purpose of storing the sets $C_{\delta,\epsilon}$, this shows that it is enough to save one out of 32 (resp. 16) elements in the $C_{\delta,\epsilon}$ tables. This results in a reduction of required disk space by a factor of $(16 + 255 * 32)/256 \approx 32$.

The second approach to save storage space is based on the following observation: an element $(k_0, k_1, k_2, k_3)$ is in $C_{\delta,\epsilon}$, if and only if $(k_0 + a, k_1 + a, k_2 + a, k_3 + a) \in C_{\delta + a,\epsilon + a}$. Thus, every set $C_{\delta,\epsilon}$ can be easily computed from the set $C_{\delta + a,\epsilon + a}$. This shows that it is enough to store for all $\delta_0 \in F_2^8$ the set $C_{\delta_0,0}$.

Combining these two approaches reduces the required disk space by a factor of approx. $128 * 32 = 2^{12}$, and hence we only need approximately 540 megabytes which is no problem on today’s PC. As a matter of fact, the sets $C_{\delta + a,\epsilon + a}$ will fit on a regular CD-ROM.

We analyze all mappings of an input $\delta$ to an output $b_i$ for a fixed key $(k_0, k_1, k_2, k_3)$ as independent random functions from $F_2^8$ to $F_2^8$ in rows one to four. We want to determine the expected number of encryptions until at least one collision has occurred in each of the output bytes $b_0, \ldots, b_3$.

As aforementioned, the probability that after $n$ encryptions at least one collision occurs
in a single output byte \( b_0, \cdots, b_3 \) is given by

\[
p(n) = 1 - \prod_{i=0}^{n-1} \left( 1 - \frac{i}{256} \right)
\]

For \( n = 20 \), \( p(20) = 0.5332 \geq 1/2 \), which means that on average 20 encryptions are required in order to detect a collision. In the optimized attack, we want to determine the average number of required encryptions such that a collision has occurred independently in each of the four outputs \( b_0, \cdots, b_3 \). Therefore, we have to compute the minimum value \( n \) such that \( p(n) \geq (1/2)^{1/4} \). Solving this inequality for the minimum number of required encryptions, we obtain \( n = 31 \). Hence, after an average of 31 encryptions we will get at least one collision in each of the four output bytes of the mix column transformation.

Every collision \((\delta, \epsilon)\) will yield possible key candidates \((k_0, k_1, k_2, k_3)\), which can be looked up in the stored tables \( C_{\delta+\epsilon,0} \). Simulated attacks showed that every new collision decreases the intersection of all key candidate sets by approximately 8 bit. As a result, we are able to determine the entire 32-bit key \((k_0, k_1, k_2, k_3)\) after collisions have been detected in all four output bytes \( b_0, \cdots, b_3 \).

Furthermore, it is possible to apply the optimized attack in parallel against all four columns. If we do not only consider the values \( b_0, \cdots, b_3 \), but also the output bytes \( b_4, \cdots, b_{15} \) of the remaining columns, we have to compute the minimal value \( n \) such that \( p(n) \geq (1/2)^{1/16} \). As a result, we get \( n = 40 \), thus after an average of 40 encryptions at least one collision will be detected in each of the 16 outputs \( b_0, \cdots, b_{15} \). These values are verified by our simulations. Thus, on average we only need 40 encryptions to determine the whole 128-bit key.

### 3.3.4. Simulation and Practical Attack

As a proof of concept, the AES collision attack was simulated on a Pentium 2.4 GHz PC and results were averaged over 10,000 random keys. As stated above, whenever a collision occurs, all possible key candidates can be derived from the sets \( C_{\delta+\epsilon,0} \) and every further collision will provide an additional set of key candidates. The intersection of all sets of key candidates must then contain the real key. As shown in Table 3.8, our simulations made clear that the number of key candidates in the intersection decreases by approximately 8 bit with each new collision. In order to check the practicability of the attack, an 8051 based microcontroller running an assembly implementation of AES without countermeasures was successfully compromised using the proposed collision attack. In our experiments, the microcontroller was running at a clock frequency of 12 MHz. At this frequency it takes about 3.017 ms to encrypt a 128-bit plaintext with a 128-bit key\(^{17}\). A host PC sent chosen plaintexts to the microcontroller and thus triggered

\(^{17}\) using on-the-fly key scheduling
3.3 Internal Collisions in AES

<table>
<thead>
<tr>
<th>no. of collisions in ( b_0, b_1, b_2 ) and ( b_3 )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>no. of key candidates</td>
<td>( 2^{32} )</td>
<td>( 16,777,114 \approx 2^{21} )</td>
<td>( 65492 \approx 2^{16} )</td>
<td>( 256.6 \approx 2^{8} )</td>
<td>( 1.1 \approx 2^{0} )</td>
</tr>
</tbody>
</table>

Table 3.8.: Average no. of key candidates after one or more collisions have occurred.

new encryptions. In order to measure the power consumption of the microcontroller a small shunt resistor (\( R_s = 39 \Omega \)) was put in series between the ground pad of the microcontroller and the ground connection of the power supply. Moreover, we replaced the original voltage source of the microcontroller with a low-noise voltage source to minimize noise superimposed by the source.

A digital oscilloscope was used to sample the voltage over the shunt resistor. We focused on collisions \( S(k_{22}) + S(k_{33}) = S(\delta + k_{22}) + S(\delta + k_{33}) \) in output byte \( b_{00} \) of the mix column transformation in the first round. Our main interest was to find out which measurement costs (sampling frequency and no. of averagings per encryption) are required to detect such a collision. Within the 8051 AES implementation the following assembly instructions in round two were directly affected by a collision in byte \( b_{00} \):

\[
\begin{align*}
\text{mov} & \quad \text{a, 30h} \quad ; (1) \text{ Read round 1 mix column output byte } b_{00} \\
\text{xrl} & \quad \text{a, 40h} \quad ; (1) \text{ X-Or } b_{00} \text{ with round 2 key byte } k_{00} \\
\text{movc} & \quad \text{a, @a+dpitr} \quad ; (2) \text{ S-box lookup} \\
\text{mov} & \quad \text{30h, a} \quad ; (1) \text{ Write back the S-box output value}
\end{align*}
\]

The number of machine cycles per instruction is given in parentheses in the remarks following the assembly instructions. Since the microcontroller is clocked at 12 MHz which corresponds to a machine cycle length of 1 \( \mu s \), this instruction sequence lasts about 5 \( \mu s \). We began our experiments at a sampling rate of 500 MHz and one single measurement per encryption, i.e. no averaging of power traces was applied. In order to examine collisions, plaintext bytes \( p_{22} = p_{33} = \delta \) were varied from \( \delta = 1...255 \) and compared with the reference trace at \( p_{22} = p_{33} = 0 \) based on the least-squares method.

\[
R[\delta] = \left( \sum_{t=t_0}^{t_0+N-1} (p(t, 0) - p(t, \delta))^2 \right)^{-1}
\]

At a sampling rate of 500 MHz the number of sampling points \( N \) is 2500. Figure 3.11 shows the deviation \( R[\delta] \) of power traces for \( \delta = 1...255 \) from the reference trace with \( \delta = 0 \). Our AES implementation used the key bytes \( k_{22} = 21 \) and \( k_{33} = 60 \), therefore we expected a distinct peak at \( \delta = k_{22} \oplus k_{33} = 41 \) as shown in Figure 3.11. It is interesting to note that no averaging of power traces was applied. Therefore, we argue that -depending
on the target implementation and measurement setup- it is possible to break the entire 128-bit key with as few as 40 measurements\textsuperscript{18}.

![Power Trace Graph](image)

Figure 3.11.: Deviation of power traces with $\delta = 1 \ldots 255$ from the reference trace with $\delta = 0$.

We also investigated other signal analysis methods such as the computation of the Pearson correlation coefficient [MS00] and continuous wavelet analysis in order to detect internal collisions. We concluded that computation of the Pearson correlation coefficient does only seem to be an appropriate method when focussing on very particular instances of time within a machine cycle, e.g. when bits on the data or address bus are switched, e.g. during clock cycle transitions. We achieved a very good detection of collisions using wavelet analysis, however, when compared with the least-squares method, its computational costs are much higher.

3.4. **Internal Collisions in Serpent**

3.4.1. **The Serpent Algorithm**

Serpent is a symmetric block cipher, which was developed by Biham et al. [BAK98]. The authors submitted it to the AES conference [ABK99], where it was selected as one of the five AES finalists. Serpent uses a block length of 128 bits and key lengths of 128, 192 and 256 bits. In the following text, we will only consider the 128-bit key variant

\textsuperscript{18}under the assumption that the attacker knows the instances when MixColumn outputs are processed in round two
of Serpent [ABK99], however, the proposed collision attack is also applicable, if larger key sizes are used. Serpent encrypts input data in 32 rounds using 33 derived 128-bit subkeys $K_0, ..., K_{32}$. The cipher mainly consists of:

- an initial permutation IP of 128 bits
- 32 rounds consisting of three steps: an x-or addition of the corresponding subkey $K_i$, a non-linear transformation of the 128-bit block using 32 equal, bijective 4-bit S-boxes in parallel and a final linear transformation step, which computes every output bit as the x-or sum of several input bits.\(^9\)
- a final permutation FP of 128 bits

The algorithm uses eight different bijective 4-bit to 4-bit S-boxes $S_0, ..., S_7$. These are used in rounds 1 to 8, in rounds 9 to 16 and so on. Hence, the algorithm can also be described as:

\[
B_0 = IP(P) \\
B_{i+1} = R_i(B_i) \\
C = FP(B_{32})
\]

where

\[
R_i(X) = L(S_{i \mod 8}(X \oplus K_i)) \quad i = 0, ..., 30 \\
R_i(X) = S_{i \mod 8}(X \oplus K_i) \oplus K_{32} \quad i = 31
\]

All S-boxes in Serpent have been particularly chosen to be 4-bit permutations with optimized differential and linear properties [ABK99]. The S-boxes $S_0, ..., S_7$ are listed in Table 3.9.

The linear transformation, which directly follows the non-linear S-box step, bijectively maps a 128-bit input block to a 128-bit output block. Each of the 128 output bits is defined as the x-or sum of several specific input bits. Therefore, the main purpose of the linear transformation in Serpent is to increase the diffusion characteristics of the cipher [Paa04]. In Table 3.10, for every single output bit of the linear transformation step the corresponding input bits, which are x-ored, are given. Note that in each row four consecutive output bits are listed, which make up the 4-bit input of an S-box in the following round.

The 128-bit round keys $K_0, ..., K_{32}$ are derived from the main key. First the main key is expanded to a 256-bit key by adding a '1' bit to the most significant, i.e., rightmost bit and padding the rest with '0' bits. The resulting 256-bit main key is then partitioned

\(^9\)In the last round the linear transformation step is replaced by the final x-or addition with subkey $k_{32}$ in order to make the cipher symmetric with respect to the encryption and decryption mode.
Internal Collision Attacks

| S-box | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| S0    | 3  | 8  | 15 | 1  | 10 | 6  | 5  | 11 | 14 | 13 | 4  | 2  | 7  | 0  | 9  | 12 |
| S1    | 15 | 12 | 2  | 7  | 9  | 0  | 5  | 10 | 1  | 11 | 14 | 8  | 6  | 13 | 3  | 4  |
| S2    | 8  | 6  | 7  | 9  | 3  | 12 | 10 | 15 | 13 | 1  | 14 | 4  | 0  | 11 | 5  | 2  |
| S3    | 0  | 15 | 11 | 8  | 12 | 9  | 6  | 3  | 13 | 1  | 2  | 4  | 10 | 7  | 5  | 14 |
| S4    | 1  | 15 | 8  | 3  | 12 | 0  | 11 | 6  | 2  | 5  | 4  | 10 | 9  | 14 | 7  | 13 |
| S5    | 15 | 5  | 2  | 11 | 4  | 10 | 9  | 12 | 0  | 3  | 14 | 8  | 13 | 6  | 7  | 1  |
| S6    | 7  | 2  | 12 | 5  | 8  | 4  | 6  | 11 | 14 | 9  | 1  | 15 | 13 | 3  | 10 | 0  |
| S7    | 1  | 13 | 15 | 0  | 14 | 8  | 2  | 11 | 7  | 4  | 12 | 10 | 9  | 3  | 5  | 6  |

Table 3.9.: The eight non-linear, bijective 4-bit S-boxes used in Serpent.

into eight 32-bit words $w_{-8}, ..., w_{-1}$. These words $w_{-8}, ..., w_{-1}$ are then used to expand to an intermediate key consisting of 32-bit words $w_0, ..., w_{131}$ using the following affine recurrence:

$$w_i := (w_{i-8} \oplus w_{i-5} \oplus w_{i-3} \oplus w_{i-1} \oplus \phi \oplus i) << 11 \quad \text{with} \quad 0 \leq i \leq 131$$

The fixed constant $\phi$ is defined as $0x9E3779B9$. The round keys $K_0, ..., K_{32}$ are derived from the prekeys $w_i$ in the following way:

$$K_0 = S_3(w_0, w_1, w_2, w_3)$$
$$K_1 = S_2(w_4, w_5, w_6, w_7)$$
$$K_2 = S_1(w_8, w_9, w_{10}, w_{11})$$
$$K_3 = S_0(w_{12}, w_{13}, w_{14}, w_{15})$$
$$K_4 = S_7(w_{16}, w_{17}, w_{18}, w_{19})$$
$$\ldots \ldots$$
$$K_{32} = S_3(w_{128}, w_{129}, w_{130}, w_{131})$$

The decryption of a ciphertext is accomplished by replacing the S-boxes and linear transformation with their inverse counterparts and by applying the subkeys in reverse order.

### 3.4.2. Partial Collisions in the Linear Transformation

As shown in the previous sections on internal collisions in DES and AES, collisions can be detected by measuring and subsequently cross-correlating side channel traces. The
<table>
<thead>
<tr>
<th>index</th>
<th>O\x00</th>
<th>O\x01</th>
<th>O\x02</th>
<th>O\x03</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>16, 52, 56, 70, 83, 94, 105</td>
<td>72, 114, 125</td>
<td>2, 9, 15, 30, 76, 84, 126</td>
<td>36, 90, 103</td>
</tr>
<tr>
<td>0x04</td>
<td>20, 56, 60, 74, 87, 98, 109</td>
<td>1, 76, 118</td>
<td>2, 6, 13, 19, 34, 80, 88</td>
<td>40, 94, 107</td>
</tr>
<tr>
<td>0x08</td>
<td>24, 60, 64, 78, 91, 102, 113</td>
<td>5, 80, 122</td>
<td>6, 10, 17, 23, 38, 84, 92</td>
<td>44, 98, 111</td>
</tr>
<tr>
<td>0x0C</td>
<td>28, 64, 68, 82, 95, 106, 117</td>
<td>9, 84, 126</td>
<td>10, 14, 21, 27, 42, 88, 96</td>
<td>48, 102, 115</td>
</tr>
<tr>
<td>0x10</td>
<td>32, 68, 72, 86, 99, 110, 121</td>
<td>2, 13, 88</td>
<td>14, 18, 25, 31, 46, 92, 100</td>
<td>52, 106, 119</td>
</tr>
<tr>
<td>0x14</td>
<td>36, 72, 76, 90, 103, 114, 125</td>
<td>6, 17, 92</td>
<td>18, 22, 29, 35, 50, 96, 104</td>
<td>56, 110, 123</td>
</tr>
<tr>
<td>0x18</td>
<td>1, 40, 76, 80, 94, 107, 118</td>
<td>10, 21, 96</td>
<td>22, 26, 33, 39, 54, 100, 108</td>
<td>60, 114, 127</td>
</tr>
<tr>
<td>0x1C</td>
<td>5, 44, 80, 84, 98, 111, 122</td>
<td>14, 25, 100</td>
<td>26, 30, 37, 43, 58, 104, 112</td>
<td>3, 118</td>
</tr>
<tr>
<td>0x20</td>
<td>9, 48, 84, 88, 102, 115, 126</td>
<td>18, 29, 104</td>
<td>30, 34, 41, 47, 62, 108, 116</td>
<td>7, 122</td>
</tr>
<tr>
<td>0x24</td>
<td>2, 13, 52, 58, 92, 106, 119</td>
<td>22, 33, 108</td>
<td>34, 38, 45, 51, 66, 112, 120</td>
<td>11, 126</td>
</tr>
<tr>
<td>0x28</td>
<td>6, 17, 56, 92, 96, 110, 123</td>
<td>26, 37, 112</td>
<td>38, 42, 49, 55, 70, 116, 124</td>
<td>2, 15, 76</td>
</tr>
<tr>
<td>0x2C</td>
<td>10, 21, 60, 66, 100, 114, 127</td>
<td>30, 41, 116</td>
<td>0, 42, 46, 53, 59, 74, 120</td>
<td>6, 19, 80</td>
</tr>
<tr>
<td>0x30</td>
<td>3, 14, 25, 100, 104, 118</td>
<td>34, 45, 120</td>
<td>4, 46, 50, 57, 63, 78, 124</td>
<td>10, 23, 84</td>
</tr>
<tr>
<td>0x34</td>
<td>7, 18, 29, 104, 108, 122</td>
<td>38, 49, 124</td>
<td>0, 8, 50, 54, 61, 67, 82</td>
<td>14, 27, 88</td>
</tr>
<tr>
<td>0x38</td>
<td>11, 22, 33, 108, 112, 126</td>
<td>0, 42, 53</td>
<td>4, 12, 54, 58, 65, 71, 86</td>
<td>18, 31, 92</td>
</tr>
<tr>
<td>0x3C</td>
<td>2, 15, 26, 37, 76, 112, 116</td>
<td>4, 46, 57</td>
<td>8, 16, 58, 62, 69, 75, 90</td>
<td>22, 35, 96</td>
</tr>
<tr>
<td>0x40</td>
<td>6, 19, 30, 41, 80, 116, 120</td>
<td>8, 50, 61</td>
<td>12, 20, 62, 66, 73, 79, 94</td>
<td>26, 39, 100</td>
</tr>
<tr>
<td>0x44</td>
<td>10, 23, 34, 45, 84, 120, 124</td>
<td>12, 54, 65</td>
<td>16, 24, 66, 70, 77, 83, 98</td>
<td>30, 43, 104</td>
</tr>
<tr>
<td>0x48</td>
<td>0, 14, 27, 38, 49, 88, 124</td>
<td>16, 58, 69</td>
<td>20, 28, 70, 74, 81, 87, 102</td>
<td>34, 47, 108</td>
</tr>
<tr>
<td>0x4C</td>
<td>0, 4, 18, 31, 43, 52, 93</td>
<td>20, 62, 73</td>
<td>24, 32, 74, 78, 85, 91, 106</td>
<td>38, 51, 112</td>
</tr>
<tr>
<td>0x50</td>
<td>4, 8, 22, 35, 46, 57, 96</td>
<td>24, 66, 77</td>
<td>28, 36, 78, 82, 89, 95, 110</td>
<td>42, 55, 116</td>
</tr>
<tr>
<td>0x54</td>
<td>8, 12, 26, 39, 50, 61, 100</td>
<td>28, 70, 81</td>
<td>32, 40, 82, 86, 93, 99, 114</td>
<td>46, 59, 120</td>
</tr>
<tr>
<td>0x58</td>
<td>12, 16, 30, 43, 54, 65, 104</td>
<td>32, 74, 85</td>
<td>36, 90, 103, 118</td>
<td>50, 63, 124</td>
</tr>
<tr>
<td>0x5C</td>
<td>16, 20, 34, 47, 58, 69, 108</td>
<td>36, 78, 89</td>
<td>40, 94, 107, 122</td>
<td>0, 54, 67</td>
</tr>
<tr>
<td>0x60</td>
<td>20, 24, 38, 51, 62, 73, 112</td>
<td>40, 82, 93</td>
<td>44, 98, 111, 126</td>
<td>4, 58, 71</td>
</tr>
<tr>
<td>0x64</td>
<td>24, 28, 42, 55, 66, 77, 116</td>
<td>44, 86, 97</td>
<td>2, 48, 102, 115</td>
<td>8, 62, 75</td>
</tr>
<tr>
<td>0x68</td>
<td>28, 32, 46, 59, 70, 81, 120</td>
<td>48, 90, 101</td>
<td>6, 52, 106, 119</td>
<td>12, 66, 79</td>
</tr>
<tr>
<td>0x6C</td>
<td>32, 36, 50, 63, 74, 85, 124</td>
<td>52, 94, 105</td>
<td>10, 56, 110, 123</td>
<td>16, 70, 83</td>
</tr>
<tr>
<td>0x70</td>
<td>0, 36, 40, 54, 67, 78, 89</td>
<td>56, 98, 109</td>
<td>14, 60, 114, 127</td>
<td>20, 74, 87</td>
</tr>
<tr>
<td>0x74</td>
<td>4, 40, 44, 58, 71, 82, 93</td>
<td>60, 102, 113</td>
<td>3, 18, 72, 114, 118, 125</td>
<td>24, 78, 91</td>
</tr>
<tr>
<td>0x78</td>
<td>8, 44, 48, 62, 75, 86, 97</td>
<td>64, 106, 117</td>
<td>1, 7, 22, 76, 88, 122</td>
<td>28, 82, 95</td>
</tr>
<tr>
<td>0x7C</td>
<td>12, 48, 52, 66, 79, 90, 101</td>
<td>68, 110, 121</td>
<td>5, 11, 26, 80, 122, 126</td>
<td>32, 86, 99</td>
</tr>
</tbody>
</table>

Table 3.10.: The linear transformation step computes every output bit by an x-or addition of several input bits.
partial collision attack against Serpent is based on the fundamental hypothesis\textsuperscript{20} that the leakage of side channel traces cross-correlates considerably higher during round two (e.g. \( \geq 95\% \)), if a single output bit instead of many output bits of the linear transformation in round one changes. The main purpose of the linear transformation in Serpent is diffusion, i.e., every output bit depends on a high number of input bits. In Table 3.11, it is shown, how many output bits depend on each input bit. For example, if input bit 0 is flipped while all other input bits are fixed, seven output bits will change.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.11.: Number of output bits of the linear transformation in Serpent, which change, if a particular input bit changes. The indices of the 128 input bits are given in hexadecimal notation.

The collision attack against Serpent is based on the idea of causing collisions in as many output bits of the linear transformation as possible by varying several input bits at the same time. Since the linear transformation is a bijective mapping, the optimum result would be that only one out of 128 output bits changes while all remaining output bits collide. Let us assume an adversary varies the plaintext and, thus, tries to cause collisions in the output bits of the linear transformation in the first round. In order to detect these collisions he first stores a reference power trace (or possibly an electromagnetic radiation trace) corresponding to the encryption of a random plaintext \( X \) on his computer. In the following, we denote the 128-bit plaintext \( X \) and the round key \( K_0 \) as the concatenation of 4-bit blocks, i.e. \( X = x_1|x_2|...|x_{32} \) and \( K_0 = k_{0,1}|k_{0,2}|...|k_{0,32} \), respectively. Next, the adversary varies the plaintext and tries to cause collisions in as many output bits of the linear transformation of round one as possible.

For example, let us consider the case that the adversary is able to alter input bits 72 (0x48) and 125 (0x7D) of the linear transformation in round one, while all other input bits remain constant. As listed in Table 3.11, an inverted input bit 72 will affect four output bits and an inverted input bit 125 will affect three output bits. To be more accurate, Table 3.10 reveals that input bit 72 will affect the output bits 1, 16, 20 and 118, while input bit 125 will affect the output bits 1, 20 and 118. If both input bits are

---

\textsuperscript{20}The correctness of this hypothesis is shown in the partial collision attack against AES (see Section 3.3), but also in [LMV04] and [Wie03].
changed simultaneously by the adversary, only output bit 16 changes while output bits 1,20 and 118 remain constant.

\[
\begin{align*}
L[1] &= S_0(X \oplus K_0)[72] \oplus S_0(X \oplus K_0)[114] \oplus S_0(X \oplus K_0)[125] \\
L[16] &= S_0(X \oplus K_0)[32] \oplus S_0(X \oplus K_0)[68] \oplus S_0(X \oplus K_0)[72] \oplus \\
&\quad S_0(X \oplus K_0)[86] \oplus S_0(X \oplus K_0)[99] \oplus S_0(X \oplus K_0)[110] \oplus \\
&\quad S_0(X \oplus K_0)[121] \\
L[20] &= S_0(X \oplus K_0)[36] \oplus S_0(X \oplus K_0)[72] \oplus S_0(X \oplus K_0)[76] \oplus \\
&\quad S_0(X \oplus K_0)[90] \oplus S_0(X \oplus K_0)[103] \oplus S_0(X \oplus K_0)[114] \oplus \\
&\quad S_0(X \oplus K_0)[125] \\
L[118] &= S_0(X \oplus K_0)[3] \oplus S_0(X \oplus K_0)[18] \oplus S_0(X \oplus K_0)[72] \oplus \\
&\quad S_0(X \oplus K_0)[114] \oplus S_0(X \oplus K_0)[118] \oplus S_0(X \oplus K_0)[125]
\end{align*}
\]

Hence, the adversary needs to find a plaintext \( X' \), which only flips the input bits 72 and 125 of the linear transformation. This corresponds to the 4-bit S-box output differentials \( \delta_{19} = 1000|_2 = 8 \) and \( \delta_{32} = 0100|_2 = 4 \) of the 19th and 32nd S-box \( S_0 \) in round one. Thus, the plaintext \( X' \) differs from plaintext \( X \) only in the two 4-bit S-box input differentials \( \epsilon_{19} \) and \( \epsilon_{32} \) with regard to the 19th and 32nd S-box, respectively.

| \( \epsilon, \delta \) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|
| 1                   | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 2 | 0 |
| 2                   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 2 | 1 | 1 | 2 |
| 3                   | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 1 | 1 | 0 | 0 | 0 |
| 4                   | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 2 | 2 | 0 | 0 |
| 5                   | 0 | 0 | 1 | 0 | 2 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 6                   | 0 | 1 | 1 | 2 | 0 | 1 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7                   | 0 | 0 | 1 | 0 | 2 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 8                   | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 2 | 0 |
| 9                   | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 10                  | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 1 | 1 | 0 | 0 | 0 |
| 11                  | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 2 |
| 12                  | 0 | 1 | 0 | 0 | 2 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 13                  | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 14                  | 0 | 1 | 0 | 0 | 2 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 15                  | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |

Table 3.12.: Distribution of 4-bit output differentials \( \delta \) for a given 4-bit input differential \( \epsilon \) with regard to S-box \( S_0 \) of Serpent.

As shown in Table 3.12, a specific S-box output differential \( \epsilon \) can only be caused by certain input differentials \( \delta \). Hence, for a given reference plaintext \( X \), the adversary may choose to check only those plaintexts \( X' \) for an internal collision, which may result in the desired S-box output differential \( \delta \). For example, in the case of output differentials
$\delta_1 = 1000_2 = 8$ and $\delta_32 = 0100_2 = 4$, the adversary only needs to check possible S-box input differentials $\epsilon_19 = \{5, 7, 12, 13, 14, 15\}$ and $\epsilon_32 = \{2, 11, 13, 15\}$, i.e., $6 \cdot 4 = 24$ additional plaintexts $X'$. On average he will find a collision after $\frac{24}{2} = 12$ additional encryptions. For a given output differential $\delta$, the occurring input differentials $\epsilon$ are associated with particular pairs of S-box inputs $(z, z \oplus \epsilon)$. Once the adversary detects a collision for a particular pair $(\epsilon_19, \epsilon_32)$, he can derive possible S-box inputs $z_{19} = (x_{19} \oplus k_{0,19}), z'_{19} = (x_{19} \oplus \epsilon_{19} \oplus k_{0,19})$ and $z_{32} = (x_{32} \oplus k_{0,32}), z'_{32} = (x_{32} \oplus \epsilon_{32} \oplus k_{0,32})$ and, thus, the corresponding key candidates $k_{0,19}, k_{0,19} + \epsilon_{19}$ and $k_{0,32}, k_{0,32} + \epsilon_{32}$.

Another approach, which requires more encryptions but also results in more collisions, is to encrypt all S-box input differentials and cross-correlate among all measured side channel traces. In the example above, the adversary would encrypt the plaintext $X$ and, next, all possible input differentials $\epsilon_19 = \{1, ..., 15\}$ and $\epsilon_32 = \{1, ..., 15\}$, i.e., $15 \cdot 15 = 225$ additional plaintexts $X'$. This corresponds to a total of $1 + 225 = 226$ side channel traces. As shown in Table 3.12, a fixed S-box output differential $\delta$ occurs for exactly 8 different S-box input pairs. With this approach the adversary hypothesizes both keys $k_{0,19}$ and $k_{0,32}$ and cross-correlates all pairs of side channel traces, which result in the desired output differentials $\delta_{19}$ and $\delta_{32}$ for a given key hypothesis.

In order to find the optimum linear combinations of input bits, which result in a single 4-bit block change at the output of the linear transformation, several combinations were tested in a computer simulation. In these simulations the upper bound, i.e., the maximum number of input bits, which were simultaneously changed, was fixed to seven bits due to limited processing power. Moreover, the more input bits of the linear transformation have to be changed in a collision attack, the longer it will take to find the corresponding pair of plaintexts $(X, X')$. The results of the simulations are listed in Tables 3.13 to 3.18. The first and third column of each table give the index of the 4-bit output block, which is changed, if the corresponding input bits flip.

In general, an adversary will try to cause collisions by varying only two or three input bits of the linear transformation in round one in order to minimize measurement costs. The number of measurements required to cause collisions in a single output bit is greatly reduced, if the adversary knows some of the underlying secret keys already. For example, let us assume an adversary tries to flip the input bits 72 and 125 of the linear transformation first and as a result, determines possible subkeys $k_{0,18}$ and $k_{0,31}$. Next, he tries to invert input bits 72, 121 and 125. Since the keys $k_{0,18}$ and $k_{0,31}$ are already known, the adversary only needs to generate random inputs $x'_{30}$ until he detects
3.5 Internal Collisions in Kasumi

3.5.1. Brief Overview of KASUMI

The KASUMI algorithm is a variant of MISTY1 which was developed by Matsui [Mat00]. The KASUMI algorithm is the cryptographic primitive chosen by the Third Generation Partnership Project (3GPP) in the Universal Mobile Telecommunications System (UMTS) standard. Within the security architecture of the 3GPP system there are two standardized confidentiality and integrity algorithms: $f_8$ and $f_9$. Both algorithms are based on the KASUMI algorithm. Moreover, the GSM Association Security Group recently provided the new stream cipher $A5/3$, which is based on KASUMI, as well.

KASUMI is a Feistel cipher with 8 rounds, which uses a 128-bit symmetric key [Con02] to encrypt a 64-bit block. The key expansion algorithm of KASUMI uses x-or additions operations with constants and fixed rotations in order to compute the 128-bit round keys. The function $f_i$, which is embedded in every Feistel round, is non-linear but unlike DES bijective. Function $f_i$ is split into two subfunctions $FL$ and $FO$. Their order within $f_i$ depends on whether it is an even round or an odd round. If the round $i$ is 1, 3, 5 or 7, then $f_i$ is defined as

$$f_i(I) = FO(FL(I, KL_i), KO_i, KL_i)$$

If the round $i$ is 2, 4, 6 or 8, then $f_i$ is defined as

$$f_i(I) = FL(FO(I, KO_i, KL_i), KL_i)$$

The keys $KL_i$, $KO_i$ and $KL_i$ are subkeys of the 128-bit round key and their lengths are 32, 48 and 48 bits, respectively. Function $FL$, which uses the subkey $KL_i$, is non-linear and invertible. It is shown in Figure 3.12. If the 32-bit input of function $FL$ is split into two 16-bit halves $L$ and $R$ and the 32-bit subkey $KL_i$ is split into the halves $KL_{i,1}$ and $KL_{i,2}$, the output $(L' | R')$ is defined as

$$R' = R \oplus ((L \cap KL_{i,1}) <<< 1)$$

$$L' = L \oplus ((R' \cup KL_{i,2}) <<< 1)$$

a collision. This divide-and-conquer approach makes it possible to increase the number of inverted input bits step by step while minimizing the measurement costs.

<table>
<thead>
<tr>
<th>4-bit output</th>
<th>linear transform input bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>64 and 117</td>
</tr>
<tr>
<td>3</td>
<td>68 and 121</td>
</tr>
<tr>
<td>4</td>
<td>72 and 125</td>
</tr>
</tbody>
</table>

Table 3.13.: All possible combinations of two input bits which will only change one 4-bit output.
### Table 3.14: All possible combinations of three input bits which will only change one 4-bit output.

<table>
<thead>
<tr>
<th>4-bit output</th>
<th>linear transform input bits</th>
<th>4-bit output</th>
<th>linear transform input bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>0,53 and 67</td>
<td>22</td>
<td>1,3 and 118</td>
</tr>
<tr>
<td>5</td>
<td>1,15 and 76</td>
<td>25</td>
<td>2,13 and 15</td>
</tr>
<tr>
<td>19</td>
<td>4,57 and 71</td>
<td>23</td>
<td>5,7 and 122</td>
</tr>
<tr>
<td>6</td>
<td>5,19 and 80</td>
<td>26</td>
<td>6,17 and 19</td>
</tr>
<tr>
<td>20</td>
<td>8,61 and 75</td>
<td>24</td>
<td>9,11 and 126</td>
</tr>
<tr>
<td>7</td>
<td>9,23 and 84</td>
<td>27</td>
<td>10,21 and 23</td>
</tr>
<tr>
<td>21</td>
<td>12,65 and 79</td>
<td>8</td>
<td>13,27 and 88</td>
</tr>
<tr>
<td>28</td>
<td>14,25 and 27</td>
<td>22</td>
<td>16,69 and 83</td>
</tr>
<tr>
<td>9</td>
<td>17,31 and 92</td>
<td>29</td>
<td>18,29 and 31</td>
</tr>
<tr>
<td>23</td>
<td>20,73 and 87</td>
<td>10</td>
<td>21,35 and 96</td>
</tr>
<tr>
<td>30</td>
<td>22,33 and 35</td>
<td>24</td>
<td>24,77 and 91</td>
</tr>
<tr>
<td>11</td>
<td>25,39 and 100</td>
<td>31</td>
<td>26,37 and 39</td>
</tr>
<tr>
<td>25</td>
<td>28,81 and 95</td>
<td>12</td>
<td>29,43 and 104</td>
</tr>
<tr>
<td>0</td>
<td>30,41 and 43</td>
<td>26</td>
<td>32,85 and 99</td>
</tr>
<tr>
<td>13</td>
<td>33,47 and 108</td>
<td>1</td>
<td>34,45 and 47</td>
</tr>
<tr>
<td>27</td>
<td>36,89 and 103</td>
<td>14</td>
<td>37,51 and 112</td>
</tr>
<tr>
<td>2</td>
<td>38,49 and 51</td>
<td>28</td>
<td>40,93 and 107</td>
</tr>
<tr>
<td>15</td>
<td>41,55 and 116</td>
<td>3</td>
<td>42,53 and 55</td>
</tr>
<tr>
<td>29</td>
<td>44,97 and 111</td>
<td>16</td>
<td>45,59 and 120</td>
</tr>
<tr>
<td>4</td>
<td>46,57 and 59</td>
<td>30</td>
<td>48,101 and 115</td>
</tr>
<tr>
<td>17</td>
<td>49,63 and 124</td>
<td>5</td>
<td>50,61 and 63</td>
</tr>
<tr>
<td>31</td>
<td>52,105 and 119</td>
<td>6</td>
<td>54,65 and 67</td>
</tr>
<tr>
<td>0</td>
<td>56,109 and 123</td>
<td>7</td>
<td>58,69 and 71</td>
</tr>
<tr>
<td>1</td>
<td>60,113 and 127</td>
<td>8</td>
<td>62,73 and 75</td>
</tr>
<tr>
<td>29</td>
<td>64,113 and 117</td>
<td>9</td>
<td>66,77 and 79</td>
</tr>
<tr>
<td>30</td>
<td>68,117 and 121</td>
<td>10</td>
<td>70,81 and 83</td>
</tr>
<tr>
<td>31</td>
<td>72,121 and 125</td>
<td>11</td>
<td>74,85 and 87</td>
</tr>
<tr>
<td>12</td>
<td>78,89 and 91</td>
<td>13</td>
<td>82,93 and 95</td>
</tr>
<tr>
<td>14</td>
<td>86,97 and 99</td>
<td>15</td>
<td>90,101 and 103</td>
</tr>
<tr>
<td>16</td>
<td>94,105 and 107</td>
<td>17</td>
<td>98,109 and 111</td>
</tr>
<tr>
<td>18</td>
<td>102,113 and 115</td>
<td>19</td>
<td>106,117 and 119</td>
</tr>
<tr>
<td>20</td>
<td>110,121 and 123</td>
<td>21</td>
<td>114,125 and 127</td>
</tr>
</tbody>
</table>

### Table 3.15: All possible combinations of four input bits which will only change one 4-bit output.

<table>
<thead>
<tr>
<th>4-bit output</th>
<th>linear transform input bits</th>
<th>4-bit output</th>
<th>linear transform input bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>48,97,101 and 115</td>
<td>26</td>
<td>52,101,105 and 119</td>
</tr>
<tr>
<td>27</td>
<td>56,105,109 and 123</td>
<td>28</td>
<td>60,109,113 and 127</td>
</tr>
<tr>
<td>29</td>
<td>64,91,106 and 119</td>
<td>30</td>
<td>68,95,110 and 123</td>
</tr>
<tr>
<td>31</td>
<td>72,99,114 and 127</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 3.5 Internal Collisions in Kasumi

<table>
<thead>
<tr>
<th>4-bit output</th>
<th>linear transform input bits</th>
<th>4-bit output</th>
<th>linear transform input bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0.27, 42, 55 and 67</td>
<td>3</td>
<td>2.15, 27, 88 and 115</td>
</tr>
<tr>
<td>0</td>
<td>3.15, 76, 103 and 118</td>
<td>7</td>
<td>3.18, 31, 43 and 104</td>
</tr>
<tr>
<td>14</td>
<td>4.31, 46, 59 and 71</td>
<td>4</td>
<td>6.19, 31, 92 and 119</td>
</tr>
<tr>
<td>1</td>
<td>7.19, 80, 107 and 122</td>
<td>8</td>
<td>7.22, 35, 47 and 108</td>
</tr>
<tr>
<td>15</td>
<td>8.35, 50, 63 and 75</td>
<td>5</td>
<td>10.23, 35, 96 and 123</td>
</tr>
<tr>
<td>2</td>
<td>11.23, 84, 111 and 126</td>
<td>9</td>
<td>11.26, 39, 51 and 112</td>
</tr>
<tr>
<td>16</td>
<td>12.39, 54, 67 and 79</td>
<td>6</td>
<td>14.27, 39, 100 and 127</td>
</tr>
<tr>
<td>10</td>
<td>15.30, 43, 55 and 116</td>
<td>17</td>
<td>16.43, 58, 71 and 83</td>
</tr>
<tr>
<td>11</td>
<td>19.34, 47, 59 and 120</td>
<td>18</td>
<td>20.47, 62, 75 and 87</td>
</tr>
<tr>
<td>12</td>
<td>23.38, 51, 63 and 124</td>
<td>19</td>
<td>24.51, 66, 79 and 91</td>
</tr>
<tr>
<td>20</td>
<td>28.55, 70, 83 and 95</td>
<td>21</td>
<td>32.59, 74, 87 and 99</td>
</tr>
<tr>
<td>22</td>
<td>36.63, 78, 91 and 103</td>
<td>2</td>
<td>38.49, 51, 64 and 117</td>
</tr>
<tr>
<td>23</td>
<td>40.67, 82, 95 and 107</td>
<td>3</td>
<td>42.53, 55, 68 and 121</td>
</tr>
<tr>
<td>24</td>
<td>44.71, 86, 99 and 111</td>
<td>4</td>
<td>46.57, 59, 72 and 125</td>
</tr>
<tr>
<td>25</td>
<td>48.75, 90, 103 and 115</td>
<td>26</td>
<td>52.79, 94, 107 and 119</td>
</tr>
<tr>
<td>27</td>
<td>56.83, 98, 111 and 123</td>
<td>28</td>
<td>60.87, 102, 115 and 127</td>
</tr>
<tr>
<td>25</td>
<td>75.90, 97, 101 and 103</td>
<td>26</td>
<td>79.94, 101, 105 and 107</td>
</tr>
<tr>
<td>27</td>
<td>83.98, 105, 109 and 111</td>
<td>28</td>
<td>87.102, 109, 113 and 115</td>
</tr>
<tr>
<td>29</td>
<td>91.106, 113, 117 and 119</td>
<td>30</td>
<td>95.110, 117, 121 and 123</td>
</tr>
<tr>
<td>31</td>
<td>99.114, 121, 125 and 127</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.16.: All possible combinations of five input bits which will only change one 4-bit output.

<table>
<thead>
<tr>
<th>4-bit output</th>
<th>linear transform input bits</th>
<th>4-bit output</th>
<th>linear transform input bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>0.53, 67, 102, 113 and 115</td>
<td>22</td>
<td>1.3, 16, 69, 83 and 118</td>
</tr>
<tr>
<td>5</td>
<td>1.15, 50, 61, 63 and 76</td>
<td>25</td>
<td>2.13, 15, 28, 81 and 95</td>
</tr>
<tr>
<td>19</td>
<td>4.57, 71, 106, 117 and 119</td>
<td>23</td>
<td>5.7, 20, 73, 87 and 122</td>
</tr>
<tr>
<td>6</td>
<td>5.19, 54, 65, 67 and 80</td>
<td>26</td>
<td>6.17, 19, 32, 85 and 99</td>
</tr>
<tr>
<td>29</td>
<td>8.61, 75, 110, 121 and 123</td>
<td>24</td>
<td>9.11, 24, 77, 91 and 126</td>
</tr>
<tr>
<td>7</td>
<td>9.23, 58, 69, 71 and 84</td>
<td>27</td>
<td>10.21, 23, 36, 89 and 103</td>
</tr>
<tr>
<td>21</td>
<td>12.63, 79, 114, 125 and 127</td>
<td>8</td>
<td>13.2, 76, 23, 75 and 88</td>
</tr>
<tr>
<td>28</td>
<td>14.25, 23, 40, 93 and 107</td>
<td>9</td>
<td>17.31, 60, 77, 79 and 92</td>
</tr>
<tr>
<td>29</td>
<td>18.29, 31, 44, 97 and 111</td>
<td>29</td>
<td>18.29, 31, 64, 113 and 117</td>
</tr>
<tr>
<td>10</td>
<td>21.35, 70, 81, 83 and 96</td>
<td>30</td>
<td>22.35, 35, 48, 101 and 115</td>
</tr>
<tr>
<td>30</td>
<td>22.33, 35, 68, 117 and 121</td>
<td>11</td>
<td>25.39, 74, 85, 87 and 100</td>
</tr>
<tr>
<td>31</td>
<td>26.37, 39, 52, 105 and 119</td>
<td>31</td>
<td>26.37, 39, 72, 121 and 125</td>
</tr>
<tr>
<td>12</td>
<td>29.43, 78, 89, 91 and 104</td>
<td>0</td>
<td>30.41, 43, 56, 109 and 123</td>
</tr>
<tr>
<td>13</td>
<td>33.47, 82, 93, 95 and 108</td>
<td>1</td>
<td>34.45, 47, 60, 113 and 127</td>
</tr>
<tr>
<td>14</td>
<td>37.51, 86, 97, 99 and 112</td>
<td>15</td>
<td>41.59, 90, 101, 103 and 116</td>
</tr>
<tr>
<td>29</td>
<td>44.64, 97, 111, 113 and 117</td>
<td>16</td>
<td>45.59, 94, 105, 107 and 120</td>
</tr>
<tr>
<td>30</td>
<td>48.68, 101, 115, 117 and 121</td>
<td>17</td>
<td>49.63, 98, 109, 111 and 124</td>
</tr>
<tr>
<td>31</td>
<td>52.72, 105, 119, 121 and 125</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.17.: All possible combinations of six input bits which will only change one 4-bit output.
<table>
<thead>
<tr>
<th>4-bit output</th>
<th>linear transform input bits</th>
<th>4-bit output</th>
<th>linear transform input bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0.49, 53, 67, 70, 81 and 83</td>
<td>1</td>
<td>1.5, 19, 22, 33, 35 and 80</td>
</tr>
<tr>
<td>0</td>
<td>1, 15, 18, 29, 31, 76 and 125</td>
<td>25</td>
<td>2, 13, 15, 48, 97, 101 and 115</td>
</tr>
<tr>
<td>3</td>
<td>2, 15, 27, 68, 88, 115 and 121</td>
<td>14</td>
<td>4, 53, 57, 71, 74, 85 and 87</td>
</tr>
<tr>
<td>2</td>
<td>5, 9, 23, 26, 37, 39 and 84</td>
<td>26</td>
<td>6, 17, 19, 52, 101, 105 and 119</td>
</tr>
<tr>
<td>4</td>
<td>6, 19, 31, 72, 92, 119 and 125</td>
<td>15</td>
<td>8, 57, 61, 75, 78, 89 and 91</td>
</tr>
<tr>
<td>3</td>
<td>9, 13, 27, 30, 41, 43 and 88</td>
<td>27</td>
<td>10, 21, 23, 56, 105, 109 and 123</td>
</tr>
<tr>
<td>2</td>
<td>11, 23, 64, 84, 111, 117 and 126</td>
<td>16</td>
<td>12, 61, 65, 79, 82, 93 and 95</td>
</tr>
<tr>
<td>4</td>
<td>13, 17, 31, 34, 45, 47 and 92</td>
<td>28</td>
<td>14, 25, 27, 60, 109, 113 and 127</td>
</tr>
<tr>
<td>17</td>
<td>16, 65, 69, 83, 86, 97 and 99</td>
<td>5</td>
<td>17, 21, 35, 38, 49, 51 and 96</td>
</tr>
<tr>
<td>29</td>
<td>18, 29, 31, 64, 91, 106 and 119</td>
<td>18</td>
<td>20, 69, 73, 87, 90, 101 and 103</td>
</tr>
<tr>
<td>6</td>
<td>21, 25, 39, 42, 53, 55 and 100</td>
<td>30</td>
<td>22, 33, 35, 68, 95, 110 and 123</td>
</tr>
<tr>
<td>19</td>
<td>24, 73, 77, 91, 94, 105 and 107</td>
<td>7</td>
<td>25, 29, 43, 46, 57, 59 and 104</td>
</tr>
<tr>
<td>31</td>
<td>26, 37, 39, 72, 99, 114 and 127</td>
<td>25</td>
<td>28, 48, 81, 95, 97, 101 and 115</td>
</tr>
<tr>
<td>20</td>
<td>28, 77, 81, 95, 98, 109 and 111</td>
<td>8</td>
<td>29, 33, 47, 50, 61, 63 and 108</td>
</tr>
<tr>
<td>26</td>
<td>32, 52, 85, 99, 101, 105 and 119</td>
<td>21</td>
<td>32, 81, 85, 99, 102, 113 and 115</td>
</tr>
<tr>
<td>9</td>
<td>33, 37, 51, 54, 65, 67 and 112</td>
<td>27</td>
<td>36, 56, 89, 103, 105, 109 and 123</td>
</tr>
<tr>
<td>22</td>
<td>36, 85, 89, 103, 106, 117 and 119</td>
<td>10</td>
<td>37, 41, 55, 58, 69, 71 and 116</td>
</tr>
<tr>
<td>28</td>
<td>40, 60, 93, 107, 109, 113 and 127</td>
<td>23</td>
<td>40, 89, 93, 107, 110, 121 and 123</td>
</tr>
<tr>
<td>11</td>
<td>41, 45, 59, 62, 73, 75 and 120</td>
<td>29</td>
<td>44, 64, 91, 97, 106, 111 and 119</td>
</tr>
<tr>
<td>24</td>
<td>44, 93, 97, 111, 114, 125 and 127</td>
<td>12</td>
<td>45, 49, 63, 66, 77, 79 and 124</td>
</tr>
<tr>
<td>30</td>
<td>48, 68, 95, 101, 110, 115 and 123</td>
<td>31</td>
<td>52, 72, 99, 105, 114, 119 and 127</td>
</tr>
</tbody>
</table>

Table 3.18.: All possible combinations of seven input bits which will only change one 4-bit output.

![Figure 3.12.: Function FL in KASUMI.](image-url)
3.5 Internal Collisions in Kasumi

Function $FO$ is a Feistel structure with three rounds. It is depicted in Figure 3.13. Its 32-bit input is split into two 16-halves $L_0$ and $R_0$ and the 48-bit keys $KO_i$ and $KI_i$ are split into the 16-bit subkeys $KO_{i,1}$, $KO_{i,2}$, $KO_{i,3}$, $KI_{i,1}$, $KI_{i,2}$ and $KI_{i,3}$. The output of each round $1 \leq j \leq 3$ is defined as

$$R_j = FI(L_{j-1} \oplus KO_{i,j}, KI_{i,j}) \oplus R_{j-1}$$
$$L_j = R_{j-1}$$

The final output of function $FO$ is $(L_3 | R_3)$. Function $FO$ embeds function $FI$, which is shown in Figure 3.14. Function $FI$ splits its 16-bit input into a 9-bit and a 7-bit part, marked as $x$ and $y$ in Figure 3.14. Two important components of $FI$ are the non-linear, bijective S-boxes: $S7$ maps a 7-bit input to a 7-bit output and $S9$ maps a 9-bit input to a 9-bit output. Two further components of $FI$ are the zero extension and truncation of data values: in order to convert a 7-bit value to a 9-bit value the extension step appends two most-significant zero bits, in order to convert a 9-bit value to a 7-bit value the truncation step discards the two most-significant bits.

![Figure 3.13: KASUMI Function $FO$.](image)

![Figure 3.14: KASUMI Function $FI$.](image)
3.5.2. Collisions in Function FL

Since function FL is bijective, it is not possible to cause a collision in the entire 32-bit output. However, partial collisions can occur in the upper or lower 16-bit halves \( L' \) and \( R' \). For example, an adversary may first encrypt the all-zero plaintext and measure the corresponding side channel trace during the first two rounds, which he stores as a reference trace on his computer. Next, he chooses one of the 16 bits of input \( L \) and sets the particular bit, while all other plaintext bits remain zero. The AND operation in function FL acts as a key bit masking operator, i.e., only if the corresponding bit in the subkey \( KL_{1,1} \) is zero, the change in \( L \) will not affect the output \( R' \). In this case a collision will occur in the output \( R' \). This collision is marked as collision 1 in Figure 3.12. It can be detected in the succeeding function F12, which is embedded in function FO in round one, by cross-correlation of the measured power trace with the reference trace.

Collisions in output \( L' \) can be caused in a similar way. The adversary chooses one of the 16 bits of input \( R \) and sets the particular bit, while all other bits remain zero. The OR operation in function FL acts as a key addition operator: only if the corresponding bit in the subkey \( KL_{1,2} \) is one, the change in \( R \) will not affect the output \( L' \). In this case a collision will occur in the output \( L' \). This collision is marked as collision 2 in Figure 3.12. It can be detected in the succeeding function F11, which is embedded in function FO in round one, by cross-correlation of the measured power trace with the reference trace.

3.5.3. Collisions in Function FO

Let us assume that the adversary has determined the entire 32-bit subkey \( KL_1 \) (see Section 3.5.2). Since function FL is bijective and \( KL_1 \) is known, the adversary is able to choose the input \((L_0|R_0)\) of function FO. In the following steps we suppose that the 32-bit input of FO is first set to zero\(^{21}\). Moreover, we will consider four different types of collisions in subfunction F11, only, which reveal the subkeys \( KO_{1,1} \) and \( KL_{1,1} \). Once the adversary knows subkeys \( KO_{1,1} \) and \( KL_{1,1} \), he can cause collisions in F12 and F13 in the same manner. In Figure 3.14, possible collisions in function F11 are marked as collision 3, collision 4, collision 5 and collision 6.

Collision 3: First, the adversary tries to adds such an input differential \( \Delta_i \) to the unknown input \( x \) of S-box S9 that an output differential \( \Delta_o \) occurs at S9, whose two most significant bits are zero. As proposed in [Wie03], the output change in the following xor gate can be compensated by adding the corresponding differential \( \Delta_o \) to the 7-bit branch \( y \) at the input of function FI.

\[
S9(x) \oplus S9(x \oplus \Delta_i) = \Delta_o \quad , \quad x, \Delta_i \in GF(2^9) \quad , \quad \Delta_o \in GF(2^7)
\]  
(3.58)

\(^{21}\)This initial input results in a reference side channel trace, which is used for collision detection in the subsequent steps of the attack.
3.5 Internal Collisions in Kasumi

The solutions $x$, which fulfill equation 3.58 for a given pair $(\Delta_i, \Delta_o)$, are listed in table 3.20. As a matter of fact, for a given pair $(\Delta_i, \Delta_o)$ the unique solution\(^{22}\) to equation 3.58 is the pair $(x, x \oplus \Delta_i)$. The idea of the attack is as follows: an adversary randomly guesses the input $x$, picks the corresponding differential pair $(\Delta_i, \Delta_o)$ given in Table 3.20 and chooses the concatenated 16-bit value $(\Delta_i|\Delta_o)$ as input $(x|y)$ of function $FO$. With side channel analysis a collision will then be detectable during the second S-box S9 lookup in function $FI11$. The probability of a collision is $p = \frac{1}{512} \approx 0.0019$, the expected number of measurements until a collision is detected is $E[M] = 256$. Once a collision has been detected, the input of the first S-box S9 must be either $x$ or $x \oplus \Delta_i$, which results in the two 9-bit key candidates $KO'_{1,1}[0...8]$ and $KO''_{1,1}[0...8]$. 

\[
KO'_{1,1}[0...8] = x \oplus L_0[0...8] \\
KO''_{1,1}[0...8] = x \oplus \Delta_i \oplus L_0[0...8]
\] (3.59) (3.60)

The correct subkey $KO_{1,1}[0...8]$ can be found by testing an additional differential pair\(^{23}\) $(\Delta'_i, \Delta'_o)$, which results in a collision for input $x$, but not $x \oplus \Delta_i$.

**Collision 4:** Once the correct subkey $KO_{1,1}[0...8]$ has been derived, the input $x$ and, thus, $S(x)$ can be arbitrarily set by the adversary. One possibility to cause a collision in the xor gate following the first S-box S7 is to add a chosen input differential $\Phi_i$ to the 7-bit input $y$ and then try to compensate the output differential $\Phi_o = S7(y) \oplus S7(y \oplus \Phi_i)$ by an S-box S9 output differential $\Delta_o[0...6] = (S9(x) \oplus S9(x \oplus \Delta_i))[0...6] = \Phi_o$. 

\[
S7(y) \oplus S7(y \oplus \Phi_i) = \Phi_o = \Delta_o[0...6] , \quad y \in GF(2^7) , \quad \epsilon, \Delta_o \in GF(2^7)
\] (3.61)

In Table 3.19, for the fixed input differential $\Phi = \{1\}$ all values of $\Phi_o$ corresponding to the input pairs $(y, y \oplus \Phi_i)$ are listed. The approach is analogous to the collision test aforementioned, i.e. with the help of side channel analysis a collision will be detectable during the second S-box S7 look-up. The probability of a collision is $p = \frac{1}{128} \approx 0.0078$, the expected number of measurements until a collision is detected is $E[M] = 64$. Once a collision has been detected, the input of S7 is either $y$ or $y \oplus \Phi_i$ according to table 3.19, which results in the two 7-bit key candidates $KO'_{1,1}[9...15]$ and $KO''_{1,1}[9...15]$. 

\[
KO'_{1,1}[9...15] = y \oplus L_0[9...15] \\
KO''_{1,1}[9...15] = y \oplus \Phi_i \oplus L_0[9...15]
\] (3.62) (3.63)

The correct subkey $KO_{1,1}[9...15]$ can be found by testing an additional differential pair $(\Phi'_i, \Phi'_o)$, which results in a collision for input $y$, but not $y \oplus \Phi_i$.

**Collisions 5 and 6:** Once the entire 16-bit key $KO_{1,1}$ has been extracted, collisions

---

\(^{22}\) Please note, however, that for a given input $x$ there exist several pairs $(\Delta_i, \Delta_o)$, which fulfill equation 3.58.

\(^{23}\) Additional differential pairs $(\Delta'_i, \Delta'_o)$ which fulfill equation 3.58 for a given input $x$ are not listed in Table 3.20 due to size constraints, however, these can be easily derived with an exhaustive search.
can be caused in the second pair of S-boxes S7 and S9 in exactly the same manner as mentioned above. The collision at the xor gate following the second S-box S9 in F111 marked as collision 5 will then be detectable during the first S-box S7 lookup in F113. In the same way, collision 6 in F111 will be detectable in the first S-box S9 lookup in F113. These two collisions will reveal the key $K_{1,1}$.

In the case of KASUMI, we showed that internal key dependent collisions can occur in the subfunctions $F_L$ and $F_I$. Since subfunctions $F_L$ and $F_I$ are bijective, it is possible to cause these collisions iteratively in a divide-and-conquer manner. This approach makes it possible to determine the entire 128-bit key, because of the linear key scheduling used in KASUMI.

<table>
<thead>
<tr>
<th>$y$</th>
<th>$y \oplus \Phi_i$</th>
<th>$\Phi_o$</th>
<th>$y$</th>
<th>$y \oplus \Phi_i$</th>
<th>$\Phi_o$</th>
<th>$y$</th>
<th>$y \oplus \Phi_i$</th>
<th>$\Phi_o$</th>
<th>$y$</th>
<th>$y \oplus \Phi_i$</th>
<th>$\Phi_o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>52</td>
<td>6</td>
<td>7</td>
<td>62</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>32</td>
<td>10</td>
<td>11</td>
<td>98</td>
<td>12</td>
<td>13</td>
<td>16</td>
<td>14</td>
<td>15</td>
<td>90</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>70</td>
<td>18</td>
<td>19</td>
<td>85</td>
<td>20</td>
<td>21</td>
<td>86</td>
<td>22</td>
<td>23</td>
<td>77</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>102</td>
<td>26</td>
<td>27</td>
<td>53</td>
<td>28</td>
<td>29</td>
<td>118</td>
<td>30</td>
<td>31</td>
<td>45</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>60</td>
<td>34</td>
<td>35</td>
<td>54</td>
<td>36</td>
<td>37</td>
<td>8</td>
<td>38</td>
<td>39</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
<td>26</td>
<td>42</td>
<td>43</td>
<td>80</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>46</td>
<td>47</td>
<td>108</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
<td>110</td>
<td>50</td>
<td>51</td>
<td>117</td>
<td>52</td>
<td>53</td>
<td>122</td>
<td>54</td>
<td>55</td>
<td>105</td>
</tr>
<tr>
<td>56</td>
<td>57</td>
<td>76</td>
<td>58</td>
<td>59</td>
<td>23</td>
<td>60</td>
<td>61</td>
<td>88</td>
<td>62</td>
<td>63</td>
<td>11</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
<td>1</td>
<td>66</td>
<td>67</td>
<td>71</td>
<td>68</td>
<td>69</td>
<td>51</td>
<td>70</td>
<td>71</td>
<td>125</td>
</tr>
<tr>
<td>72</td>
<td>73</td>
<td>21</td>
<td>74</td>
<td>75</td>
<td>19</td>
<td>76</td>
<td>77</td>
<td>39</td>
<td>78</td>
<td>79</td>
<td>41</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
<td>67</td>
<td>82</td>
<td>83</td>
<td>20</td>
<td>84</td>
<td>85</td>
<td>81</td>
<td>86</td>
<td>87</td>
<td>14</td>
</tr>
<tr>
<td>88</td>
<td>89</td>
<td>83</td>
<td>90</td>
<td>91</td>
<td>68</td>
<td>92</td>
<td>93</td>
<td>65</td>
<td>94</td>
<td>95</td>
<td>94</td>
</tr>
<tr>
<td>96</td>
<td>97</td>
<td>121</td>
<td>98</td>
<td>99</td>
<td>55</td>
<td>100</td>
<td>101</td>
<td>79</td>
<td>102</td>
<td>103</td>
<td>9</td>
</tr>
<tr>
<td>104</td>
<td>105</td>
<td>111</td>
<td>106</td>
<td>107</td>
<td>97</td>
<td>108</td>
<td>109</td>
<td>89</td>
<td>110</td>
<td>111</td>
<td>95</td>
</tr>
<tr>
<td>112</td>
<td>113</td>
<td>43</td>
<td>114</td>
<td>115</td>
<td>116</td>
<td>116</td>
<td>117</td>
<td>61</td>
<td>118</td>
<td>119</td>
<td>106</td>
</tr>
<tr>
<td>120</td>
<td>121</td>
<td>57</td>
<td>122</td>
<td>123</td>
<td>38</td>
<td>124</td>
<td>125</td>
<td>47</td>
<td>126</td>
<td>127</td>
<td>56</td>
</tr>
</tbody>
</table>

Table 3.19.: KASUMI S-box S7: For every differential pair $(\Phi_i, \Phi_o)$ with $\Phi_i = \{1\}$ there exist two solutions $(y, y \oplus \Phi_i)$ which fulfill the equation $\Phi_o = S7(y) \oplus S7(y \oplus \Phi_i)$ with $y, \Phi_i, \Phi_o \in GF(2^7)$. 

Internal Collision Attacks
3.5 Internal Collisions in Kasumi

<table>
<thead>
<tr>
<th>$x$</th>
<th>$x \oplus \Delta_i$</th>
<th>$\Delta_i$</th>
<th>$\Delta_o$</th>
<th>$x$</th>
<th>$x \oplus \Delta_i$</th>
<th>$\Delta_i$</th>
<th>$\Delta_o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>72</td>
<td>2</td>
<td>9</td>
<td>11</td>
<td>67</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>8</td>
<td>29</td>
<td>4</td>
<td>15</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>2</td>
<td>28</td>
<td>6</td>
<td>14</td>
<td>8</td>
<td>83</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>2</td>
<td>22</td>
<td>12</td>
<td>13</td>
<td>1</td>
<td>69</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>1</td>
<td>74</td>
<td>18</td>
<td>31</td>
<td>13</td>
<td>34</td>
</tr>
<tr>
<td>19</td>
<td>30</td>
<td>13</td>
<td>47</td>
<td>20</td>
<td>25</td>
<td>13</td>
<td>107</td>
</tr>
<tr>
<td>21</td>
<td>23</td>
<td>2</td>
<td>62</td>
<td>22</td>
<td>27</td>
<td>13</td>
<td>97</td>
</tr>
<tr>
<td>24</td>
<td>26</td>
<td>2</td>
<td>52</td>
<td>28</td>
<td>29</td>
<td>1</td>
<td>71</td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>1</td>
<td>94</td>
<td>34</td>
<td>47</td>
<td>13</td>
<td>121</td>
</tr>
<tr>
<td>35</td>
<td>46</td>
<td>13</td>
<td>116</td>
<td>36</td>
<td>41</td>
<td>13</td>
<td>48</td>
</tr>
<tr>
<td>37</td>
<td>38</td>
<td>3</td>
<td>17</td>
<td>39</td>
<td>42</td>
<td>13</td>
<td>55</td>
</tr>
<tr>
<td>40</td>
<td>43</td>
<td>3</td>
<td>22</td>
<td>44</td>
<td>45</td>
<td>1</td>
<td>83</td>
</tr>
<tr>
<td>48</td>
<td>49</td>
<td>1</td>
<td>92</td>
<td>50</td>
<td>57</td>
<td>11</td>
<td>51</td>
</tr>
<tr>
<td>51</td>
<td>58</td>
<td>9</td>
<td>89</td>
<td>52</td>
<td>63</td>
<td>11</td>
<td>122</td>
</tr>
<tr>
<td>53</td>
<td>54</td>
<td>3</td>
<td>49</td>
<td>55</td>
<td>62</td>
<td>9</td>
<td>26</td>
</tr>
<tr>
<td>56</td>
<td>59</td>
<td>3</td>
<td>54</td>
<td>60</td>
<td>61</td>
<td>1</td>
<td>81</td>
</tr>
<tr>
<td>64</td>
<td>65</td>
<td>1</td>
<td>96</td>
<td>66</td>
<td>74</td>
<td>8</td>
<td>69</td>
</tr>
<tr>
<td>67</td>
<td>73</td>
<td>10</td>
<td>101</td>
<td>68</td>
<td>78</td>
<td>10</td>
<td>33</td>
</tr>
<tr>
<td>69</td>
<td>70</td>
<td>3</td>
<td>71</td>
<td>71</td>
<td>79</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>72</td>
<td>75</td>
<td>3</td>
<td>64</td>
<td>76</td>
<td>77</td>
<td>1</td>
<td>109</td>
</tr>
<tr>
<td>80</td>
<td>81</td>
<td>1</td>
<td>98</td>
<td>82</td>
<td>95</td>
<td>13</td>
<td>90</td>
</tr>
<tr>
<td>83</td>
<td>94</td>
<td>13</td>
<td>87</td>
<td>84</td>
<td>89</td>
<td>13</td>
<td>19</td>
</tr>
<tr>
<td>85</td>
<td>86</td>
<td>3</td>
<td>103</td>
<td>87</td>
<td>90</td>
<td>13</td>
<td>20</td>
</tr>
<tr>
<td>88</td>
<td>91</td>
<td>3</td>
<td>96</td>
<td>92</td>
<td>93</td>
<td>1</td>
<td>111</td>
</tr>
<tr>
<td>96</td>
<td>97</td>
<td>1</td>
<td>118</td>
<td>98</td>
<td>111</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>110</td>
<td>13</td>
<td>12</td>
<td>100</td>
<td>105</td>
<td>13</td>
<td>72</td>
</tr>
<tr>
<td>101</td>
<td>103</td>
<td>2</td>
<td>116</td>
<td>102</td>
<td>107</td>
<td>13</td>
<td>66</td>
</tr>
<tr>
<td>104</td>
<td>106</td>
<td>2</td>
<td>126</td>
<td>108</td>
<td>109</td>
<td>1</td>
<td>123</td>
</tr>
<tr>
<td>112</td>
<td>113</td>
<td>1</td>
<td>116</td>
<td>114</td>
<td>123</td>
<td>9</td>
<td>41</td>
</tr>
<tr>
<td>115</td>
<td>121</td>
<td>10</td>
<td>1</td>
<td>116</td>
<td>126</td>
<td>10</td>
<td>69</td>
</tr>
<tr>
<td>117</td>
<td>119</td>
<td>2</td>
<td>86</td>
<td>118</td>
<td>127</td>
<td>9</td>
<td>106</td>
</tr>
<tr>
<td>120</td>
<td>122</td>
<td>2</td>
<td>92</td>
<td>124</td>
<td>125</td>
<td>1</td>
<td>121</td>
</tr>
</tbody>
</table>

Table 3.20: KASUMI S-box S9: For every differential pair $(\Delta_i, \Delta_o)$ there exist two solutions $(x, x \oplus \Delta_i)$ which fulfill the equation $\Delta_o = S9(x) \oplus S9(x \oplus \Delta_i)$ with $x, \Delta_i \in GF(2^9)$ and $\Delta_o \in GF(2^7)$. 
<table>
<thead>
<tr>
<th>$x$</th>
<th>$x \oplus \Delta_i$</th>
<th>$\Delta_i$</th>
<th>$\Delta_o$</th>
<th>$x$</th>
<th>$x \oplus \Delta_i$</th>
<th>$\Delta_i$</th>
<th>$\Delta_o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>129</td>
<td>1</td>
<td>89</td>
<td>130</td>
<td>134</td>
<td>4</td>
<td>43</td>
</tr>
<tr>
<td>131</td>
<td>132</td>
<td>7</td>
<td>109</td>
<td>133</td>
<td>135</td>
<td>2</td>
<td>31</td>
</tr>
<tr>
<td>136</td>
<td>138</td>
<td>2</td>
<td>21</td>
<td>137</td>
<td>142</td>
<td>7</td>
<td>41</td>
</tr>
<tr>
<td>139</td>
<td>143</td>
<td>4</td>
<td>104</td>
<td>140</td>
<td>141</td>
<td>1</td>
<td>84</td>
</tr>
<tr>
<td>144</td>
<td>145</td>
<td>1</td>
<td>91</td>
<td>146</td>
<td>150</td>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>147</td>
<td>148</td>
<td>7</td>
<td>69</td>
<td>149</td>
<td>151</td>
<td>2</td>
<td>61</td>
</tr>
<tr>
<td>152</td>
<td>154</td>
<td>2</td>
<td>55</td>
<td>153</td>
<td>158</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>155</td>
<td>159</td>
<td>4</td>
<td>96</td>
<td>156</td>
<td>157</td>
<td>1</td>
<td>86</td>
</tr>
<tr>
<td>160</td>
<td>161</td>
<td>1</td>
<td>79</td>
<td>162</td>
<td>167</td>
<td>5</td>
<td>54</td>
</tr>
<tr>
<td>163</td>
<td>164</td>
<td>7</td>
<td>122</td>
<td>165</td>
<td>166</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>168</td>
<td>171</td>
<td>3</td>
<td>4</td>
<td>169</td>
<td>174</td>
<td>7</td>
<td>62</td>
</tr>
<tr>
<td>170</td>
<td>172</td>
<td>5</td>
<td>120</td>
<td>172</td>
<td>173</td>
<td>1</td>
<td>66</td>
</tr>
<tr>
<td>176</td>
<td>177</td>
<td>1</td>
<td>77</td>
<td>178</td>
<td>183</td>
<td>5</td>
<td>60</td>
</tr>
<tr>
<td>179</td>
<td>180</td>
<td>7</td>
<td>82</td>
<td>181</td>
<td>182</td>
<td>3</td>
<td>35</td>
</tr>
<tr>
<td>184</td>
<td>187</td>
<td>3</td>
<td>36</td>
<td>185</td>
<td>190</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>186</td>
<td>191</td>
<td>5</td>
<td>114</td>
<td>188</td>
<td>189</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>192</td>
<td>193</td>
<td>1</td>
<td>113</td>
<td>194</td>
<td>196</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>195</td>
<td>199</td>
<td>4</td>
<td>46</td>
<td>197</td>
<td>198</td>
<td>3</td>
<td>85</td>
</tr>
<tr>
<td>200</td>
<td>203</td>
<td>3</td>
<td>82</td>
<td>201</td>
<td>207</td>
<td>6</td>
<td>67</td>
</tr>
<tr>
<td>202</td>
<td>206</td>
<td>4</td>
<td>109</td>
<td>204</td>
<td>205</td>
<td>1</td>
<td>124</td>
</tr>
<tr>
<td>208</td>
<td>209</td>
<td>1</td>
<td>115</td>
<td>210</td>
<td>212</td>
<td>6</td>
<td>32</td>
</tr>
<tr>
<td>211</td>
<td>215</td>
<td>4</td>
<td>38</td>
<td>213</td>
<td>214</td>
<td>3</td>
<td>117</td>
</tr>
<tr>
<td>216</td>
<td>219</td>
<td>3</td>
<td>114</td>
<td>217</td>
<td>223</td>
<td>6</td>
<td>105</td>
</tr>
<tr>
<td>218</td>
<td>222</td>
<td>4</td>
<td>101</td>
<td>220</td>
<td>221</td>
<td>1</td>
<td>126</td>
</tr>
<tr>
<td>224</td>
<td>225</td>
<td>1</td>
<td>103</td>
<td>226</td>
<td>228</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>227</td>
<td>230</td>
<td>5</td>
<td>27</td>
<td>229</td>
<td>231</td>
<td>2</td>
<td>119</td>
</tr>
<tr>
<td>232</td>
<td>234</td>
<td>2</td>
<td>125</td>
<td>233</td>
<td>239</td>
<td>6</td>
<td>66</td>
</tr>
<tr>
<td>235</td>
<td>238</td>
<td>5</td>
<td>85</td>
<td>236</td>
<td>237</td>
<td>1</td>
<td>106</td>
</tr>
<tr>
<td>240</td>
<td>241</td>
<td>1</td>
<td>101</td>
<td>242</td>
<td>244</td>
<td>6</td>
<td>33</td>
</tr>
<tr>
<td>243</td>
<td>246</td>
<td>5</td>
<td>17</td>
<td>245</td>
<td>247</td>
<td>2</td>
<td>85</td>
</tr>
<tr>
<td>248</td>
<td>250</td>
<td>2</td>
<td>95</td>
<td>249</td>
<td>255</td>
<td>6</td>
<td>104</td>
</tr>
<tr>
<td>251</td>
<td>254</td>
<td>5</td>
<td>95</td>
<td>252</td>
<td>253</td>
<td>1</td>
<td>104</td>
</tr>
<tr>
<td>256</td>
<td>257</td>
<td>1</td>
<td>68</td>
<td>258</td>
<td>260</td>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>259</td>
<td>262</td>
<td>5</td>
<td>110</td>
<td>261</td>
<td>263</td>
<td>2</td>
<td>68</td>
</tr>
<tr>
<td>264</td>
<td>266</td>
<td>2</td>
<td>78</td>
<td>265</td>
<td>271</td>
<td>6</td>
<td>39</td>
</tr>
<tr>
<td>267</td>
<td>270</td>
<td>5</td>
<td>32</td>
<td>268</td>
<td>269</td>
<td>1</td>
<td>73</td>
</tr>
<tr>
<td>272</td>
<td>273</td>
<td>1</td>
<td>70</td>
<td>274</td>
<td>276</td>
<td>6</td>
<td>68</td>
</tr>
<tr>
<td>275</td>
<td>278</td>
<td>5</td>
<td>100</td>
<td>277</td>
<td>279</td>
<td>2</td>
<td>102</td>
</tr>
<tr>
<td>280</td>
<td>282</td>
<td>2</td>
<td>108</td>
<td>281</td>
<td>287</td>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>283</td>
<td>286</td>
<td>5</td>
<td>42</td>
<td>284</td>
<td>285</td>
<td>1</td>
<td>75</td>
</tr>
<tr>
<td>288</td>
<td>289</td>
<td>1</td>
<td>82</td>
<td>290</td>
<td>292</td>
<td>6</td>
<td>111</td>
</tr>
<tr>
<td>291</td>
<td>295</td>
<td>4</td>
<td>120</td>
<td>293</td>
<td>294</td>
<td>3</td>
<td>69</td>
</tr>
<tr>
<td>296</td>
<td>299</td>
<td>3</td>
<td>66</td>
<td>297</td>
<td>303</td>
<td>6</td>
<td>38</td>
</tr>
<tr>
<td>298</td>
<td>302</td>
<td>4</td>
<td>59</td>
<td>300</td>
<td>301</td>
<td>1</td>
<td>95</td>
</tr>
<tr>
<td>304</td>
<td>305</td>
<td>1</td>
<td>80</td>
<td>306</td>
<td>308</td>
<td>6</td>
<td>69</td>
</tr>
<tr>
<td>307</td>
<td>311</td>
<td>4</td>
<td>112</td>
<td>309</td>
<td>310</td>
<td>3</td>
<td>101</td>
</tr>
<tr>
<td>312</td>
<td>315</td>
<td>3</td>
<td>98</td>
<td>313</td>
<td>319</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>314</td>
<td>318</td>
<td>4</td>
<td>51</td>
<td>316</td>
<td>317</td>
<td>1</td>
<td>93</td>
</tr>
</tbody>
</table>

Table 3.21.: KASUMI S-box $S9$: input differentials $\Delta_i$ and output differentials $\Delta_o = S9(x) \oplus S9(x \oplus \Delta_i)$, $x \in \{256, ..., 512\}$. 
### Table 3.22: KASUMI S-box S9: input differentials $\Delta_i$ and output differentials $\Delta_o = S9(x) \oplus S9(x \oplus \Delta_i)$, $x \in \{256, \ldots, 512\}$.

<table>
<thead>
<tr>
<th>$x$</th>
<th>$x \oplus \Delta_i$</th>
<th>$\Delta_i$</th>
<th>$\Delta_o$</th>
<th>$x$</th>
<th>$x \oplus \Delta_i$</th>
<th>$\Delta_i$</th>
<th>$\Delta_o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>320</td>
<td>321</td>
<td>1</td>
<td>108</td>
<td>322</td>
<td>327</td>
<td>5</td>
<td>67</td>
</tr>
<tr>
<td>323</td>
<td>324</td>
<td>7</td>
<td>60</td>
<td>325</td>
<td>326</td>
<td>3</td>
<td>19</td>
</tr>
<tr>
<td>328</td>
<td>331</td>
<td>3</td>
<td>20</td>
<td>329</td>
<td>334</td>
<td>7</td>
<td>120</td>
</tr>
<tr>
<td>330</td>
<td>335</td>
<td>5</td>
<td>13</td>
<td>332</td>
<td>333</td>
<td>1</td>
<td>97</td>
</tr>
<tr>
<td>336</td>
<td>337</td>
<td>1</td>
<td>110</td>
<td>338</td>
<td>343</td>
<td>5</td>
<td>73</td>
</tr>
<tr>
<td>339</td>
<td>340</td>
<td>7</td>
<td>20</td>
<td>341</td>
<td>342</td>
<td>3</td>
<td>51</td>
</tr>
<tr>
<td>344</td>
<td>347</td>
<td>3</td>
<td>52</td>
<td>345</td>
<td>350</td>
<td>7</td>
<td>80</td>
</tr>
<tr>
<td>346</td>
<td>351</td>
<td>5</td>
<td>7</td>
<td>348</td>
<td>349</td>
<td>1</td>
<td>99</td>
</tr>
<tr>
<td>352</td>
<td>353</td>
<td>1</td>
<td>122</td>
<td>354</td>
<td>358</td>
<td>4</td>
<td>125</td>
</tr>
<tr>
<td>355</td>
<td>356</td>
<td>7</td>
<td>43</td>
<td>357</td>
<td>359</td>
<td>2</td>
<td>44</td>
</tr>
<tr>
<td>360</td>
<td>362</td>
<td>2</td>
<td>38</td>
<td>361</td>
<td>366</td>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>363</td>
<td>367</td>
<td>4</td>
<td>62</td>
<td>364</td>
<td>365</td>
<td>1</td>
<td>119</td>
</tr>
<tr>
<td>368</td>
<td>369</td>
<td>1</td>
<td>120</td>
<td>370</td>
<td>374</td>
<td>4</td>
<td>117</td>
</tr>
<tr>
<td>371</td>
<td>372</td>
<td>7</td>
<td>3</td>
<td>373</td>
<td>375</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>376</td>
<td>378</td>
<td>2</td>
<td>4</td>
<td>377</td>
<td>382</td>
<td>7</td>
<td>71</td>
</tr>
<tr>
<td>379</td>
<td>383</td>
<td>4</td>
<td>54</td>
<td>380</td>
<td>381</td>
<td>1</td>
<td>117</td>
</tr>
<tr>
<td>384</td>
<td>385</td>
<td>1</td>
<td>85</td>
<td>386</td>
<td>398</td>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>387</td>
<td>399</td>
<td>12</td>
<td>21</td>
<td>388</td>
<td>395</td>
<td>15</td>
<td>68</td>
</tr>
<tr>
<td>389</td>
<td>391</td>
<td>2</td>
<td>71</td>
<td>390</td>
<td>393</td>
<td>15</td>
<td>78</td>
</tr>
<tr>
<td>392</td>
<td>394</td>
<td>2</td>
<td>77</td>
<td>396</td>
<td>397</td>
<td>1</td>
<td>88</td>
</tr>
<tr>
<td>400</td>
<td>401</td>
<td>1</td>
<td>87</td>
<td>402</td>
<td>409</td>
<td>11</td>
<td>81</td>
</tr>
<tr>
<td>403</td>
<td>411</td>
<td>8</td>
<td>105</td>
<td>404</td>
<td>415</td>
<td>11</td>
<td>24</td>
</tr>
<tr>
<td>405</td>
<td>407</td>
<td>2</td>
<td>101</td>
<td>406</td>
<td>414</td>
<td>8</td>
<td>39</td>
</tr>
<tr>
<td>408</td>
<td>410</td>
<td>2</td>
<td>111</td>
<td>412</td>
<td>413</td>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>416</td>
<td>417</td>
<td>1</td>
<td>67</td>
<td>418</td>
<td>425</td>
<td>11</td>
<td>33</td>
</tr>
<tr>
<td>419</td>
<td>426</td>
<td>9</td>
<td>50</td>
<td>420</td>
<td>431</td>
<td>11</td>
<td>104</td>
</tr>
<tr>
<td>421</td>
<td>422</td>
<td>3</td>
<td>87</td>
<td>423</td>
<td>430</td>
<td>9</td>
<td>113</td>
</tr>
<tr>
<td>424</td>
<td>427</td>
<td>3</td>
<td>80</td>
<td>428</td>
<td>429</td>
<td>1</td>
<td>78</td>
</tr>
<tr>
<td>432</td>
<td>433</td>
<td>1</td>
<td>65</td>
<td>434</td>
<td>446</td>
<td>12</td>
<td>87</td>
</tr>
<tr>
<td>435</td>
<td>447</td>
<td>12</td>
<td>90</td>
<td>436</td>
<td>442</td>
<td>14</td>
<td>34</td>
</tr>
<tr>
<td>437</td>
<td>438</td>
<td>3</td>
<td>119</td>
<td>439</td>
<td>441</td>
<td>14</td>
<td>37</td>
</tr>
<tr>
<td>440</td>
<td>443</td>
<td>3</td>
<td>112</td>
<td>444</td>
<td>445</td>
<td>1</td>
<td>76</td>
</tr>
<tr>
<td>448</td>
<td>449</td>
<td>1</td>
<td>125</td>
<td>450</td>
<td>462</td>
<td>12</td>
<td>72</td>
</tr>
<tr>
<td>451</td>
<td>463</td>
<td>12</td>
<td>69</td>
<td>452</td>
<td>458</td>
<td>14</td>
<td>119</td>
</tr>
<tr>
<td>453</td>
<td>454</td>
<td>3</td>
<td>1</td>
<td>455</td>
<td>457</td>
<td>14</td>
<td>112</td>
</tr>
<tr>
<td>456</td>
<td>459</td>
<td>3</td>
<td>6</td>
<td>460</td>
<td>461</td>
<td>1</td>
<td>112</td>
</tr>
<tr>
<td>464</td>
<td>465</td>
<td>1</td>
<td>127</td>
<td>466</td>
<td>474</td>
<td>8</td>
<td>49</td>
</tr>
<tr>
<td>467</td>
<td>473</td>
<td>10</td>
<td>104</td>
<td>468</td>
<td>478</td>
<td>10</td>
<td>44</td>
</tr>
<tr>
<td>469</td>
<td>470</td>
<td>3</td>
<td>33</td>
<td>471</td>
<td>479</td>
<td>8</td>
<td>127</td>
</tr>
<tr>
<td>472</td>
<td>475</td>
<td>3</td>
<td>38</td>
<td>476</td>
<td>477</td>
<td>1</td>
<td>114</td>
</tr>
<tr>
<td>480</td>
<td>481</td>
<td>1</td>
<td>107</td>
<td>482</td>
<td>491</td>
<td>9</td>
<td>66</td>
</tr>
<tr>
<td>483</td>
<td>489</td>
<td>10</td>
<td>12</td>
<td>484</td>
<td>494</td>
<td>10</td>
<td>72</td>
</tr>
<tr>
<td>485</td>
<td>487</td>
<td>2</td>
<td>47</td>
<td>486</td>
<td>495</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>488</td>
<td>490</td>
<td>2</td>
<td>37</td>
<td>492</td>
<td>493</td>
<td>1</td>
<td>102</td>
</tr>
<tr>
<td>496</td>
<td>497</td>
<td>1</td>
<td>105</td>
<td>498</td>
<td>510</td>
<td>12</td>
<td>7</td>
</tr>
<tr>
<td>499</td>
<td>511</td>
<td>12</td>
<td>10</td>
<td>500</td>
<td>507</td>
<td>15</td>
<td>45</td>
</tr>
<tr>
<td>501</td>
<td>503</td>
<td>2</td>
<td>13</td>
<td>502</td>
<td>505</td>
<td>15</td>
<td>39</td>
</tr>
<tr>
<td>504</td>
<td>506</td>
<td>2</td>
<td>7</td>
<td>508</td>
<td>509</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>
4. Masking Strategies for AES

In this chapter, various masking strategies are discussed in order to protect AES software implementations against first and higher order side channel attacks. The target device used mainly in this chapter for practical experiments is the Atmel AVR microcontroller, however, other typical microprocessors, such as the 8051, could have been chosen, as well. We decided to use the AVR microcontroller, because it is based on a RISC architecture and allows an easy and flexible implementation of different variants of the AES using available programming tools. The development of secure software implementations of ciphers which can not rely on hardware countermeasures in order to resist side channel attacks represents a continous challenge.

First, we will discuss previous works on the masking countermeasure in Section 4.1. Then, in Section 4.2, the AVR architecture is briefly explained. Moreover, performance details of a speed-optimized reference AVR AES implementation, which does not use any countermeasures, are presented. Parts of this section were published in 2004 at the International Conference on Information Technology (ITCC) [SP04]. In Section 4.3, an efficient masking scheme which is based on inversions in the composite field is presented. We present the performance figures of various AES assembly implementations which use this countermeasure to thwart first order side channel attacks. Parts of this section were published in 2005 at the Workshop on Information Security Applications (WISA) [OS05]. Finally, in Section 4.4, we discuss the theoretical background of higher order side channel attacks and propose various masking strategies which lead to higher order resistant AES software implementations. We present the performance figures of AES assembly implementations which are resistant against DPA attacks of various orders. Parts of this work were published at the RSA-CT 2006 conference [SP06].

4.1. Previous Work

The symmetric key block cipher Rijndael, which was developed by Joan Daemen and Vincent Rijmen, was selected in October 2000 by the U.S. National Institute of Standards and Technology (NIST) as the Advanced Encryption Standard (AES). The AES is the worldwide de-facto standard for symmetric encryption [DR02]. Therefore, it is very likely
that it will be used for many different purposes ranging from high-performance applications such as video stream encryption to low-cost (low memory, low power consumption) implementations on smart cards. Especially in the case of software implementations for smart cards limited memory (ROM, RAM, XRAM) poses a challenging constraint for implementors. Even worse, side channel attacks based on differential power analysis (DPA) [KJ99, MDS99] and its several variants such as higher order differential power analysis (HODPA) [Mes00b, AG03] cause considerable effort to come up with efficient yet secure implementations which do not succumb to these attacks. Hence, a lot of effort has been devoted in the past years to the development of efficient countermeasures for ciphers, such as the AES, against side channel attacks. Especially the so-called random masking technique has been suggested many times [Mes00a, CC00, AG01, GT02] but also algebraic techniques to protect AES implementations against side channel attacks have been proposed in various publications [BGK04, OS05, CG05, RS05].

The principle of the masking countermeasure is as follows: the input of a cipher is blinded with random masks, which diffuse and propagate during the execution of the cipher. As a result, the side channel leakage of all intermediate, key-dependent variables, which are processed by the cipher, does not correlate with the corresponding unmasked variables and, thus, side channel attacks are effectively thwarted. The most important step is the final mask correction, which removes the evolved masks from the output of the cipher. While it is simple to reproduce the propagation of masks throughout linear functions in a cipher, non-linear functions, such as substitution boxes, require considerable effort when it comes to the mask correction step. In a typical software implementation, S-box operations are implemented as a table look-ups. Hence, for an input value $x$ of an S-box operation, the output is derived as $y = S(x)$. For example, there are 16 bytes in the AES state, and 16 look-up operations to the same table have to be performed in one encryption round (not taking the key schedule into account).

When we mask the S-box output with a value $m$ (the mask), i.e., when we add a random value $m'$ (the mask) to its input, we have to re-compute the table $S$ such that $y = S(x + m') + m$, where $x + m'$ denotes the masked input. Hence, we need a modified S-box $S^*(\cdot)$ such that

$$S^*(x + m') = S(x) + m = y + m$$

The $S^*(\cdot)$ table for the input mask $m$ and output mask $m'$ (for simplicity, we often choose $m$ to be equal to $m'$) is calculated according to Algorithm 1 and was originally published by Messerges et al. in [Mes00a]. The exclusive-or (short: x-or) operation is denoted by $+$, since all bytes processed by AES represent elements in the finite field $GF(2^8)$. If more than one mask $m$ is used, more $S^*(\cdot)$ tables need to be computed. For example, when using 16 different masks $m$ for the 16 plaintext bytes, 16 tables are needed. As stated in [GT02], the usage of the same mask for all 16 S-boxes represents a serious threat, because intermediate variables (e.g., the S-box outputs) are masked with the same mask and their mutual correlation can give rise to second order DPA attacks. We will discuss this particular second order attack in more detail in Section 4.4.5.
4.1 Previous Work

Algorithm 1 Computation of the Masked AES S-box as proposed by Messerges et al. in [Mes00a]

Require: $m$, $m'$
Ensure: $S^*(x \oplus m') = S(x) \oplus m$
1: for $i = 0$ to $255$ do
2: \[ S^*(i \oplus m') = S(i) \oplus m \]
3: end for
4: Return ($S^*$)

For every mask $m$, a masked table needs to be computed. There are several strategies an implementor can follow. Either all 256 masked tables are stored in ROM, or, only $t$ tables for the $t$ 8-bit masks are precomputed at the beginning of the AES algorithm and stored in RAM. Another option is to compute the masked table on the fly whenever it is needed during the encryption algorithm. In practice, the latter method is the most attractive one, because it gives the best tradeoff between the amount of memory and the number of operations. Remember that in case of AES the size of one S-box table is 256 bytes. Counting the number of operations for this algorithm for $t$ masks shows that in total an amount of $2 \times t \times 256$ table look-ups read/write operations and $2 \times t \times 256$ XOR operations are needed. In total, 256 bytes of ROM and $t \times 256$ bytes of RAM are used. In typical AES implementations, a separate mask for each byte in the state matrix would be used. That amounts then to 8192 table look-ups, 4352 bytes of RAM and 8192 XOR operations.

Also, many algorithmic countermeasures have been proposed for the AES algorithm, see [AG01], [GT02], [TSG02], [TK04], [BGK04] and [OMPR05]. They are all based on masking the intermediate value, i.e., adding a random number (the mask) to the intermediate AES values. However two of them, [AG01] and [TSG02], are both susceptible to a certain type of (first order) differential side channel attack, the zero-value attack. The latter one has turned out to be vulnerable even to standard differential side channel attacks [ABG04].

The countermeasure presented in [GT02] leads to very costly implementations. This is due to the fact that in order to circumvent the zero-value problem, the authors propose to embed the inversion operation (which is part of S-box) into a larger algebraic structure such that the zero-value is mapped to different non-zero values. Although this construction is mathematically elegant, implementations thereof, especially on 8-bit platforms, are not.

The countermeasure presented in [TK04] uses pre-computed discrete logarithm and exponentiation tables to realize the S-box operation (i.e., the inversion operation that is part of the mathematical description of S-box). This approach is based on the fact that a non-zero element in a finite field can be inverted by computing the logarithm of the
element to a particular base\(^1\) and exponentiating the base again with the negated logarithm. The inversion of the zero element has to be carefully taken into account by using a conditional check, e.g. the authors suggest to manipulate the discrete logarithm and exponentiation tables in such a way that the zero element is inverted correctly to itself. Unfortunately, we believe that this approach has a flaw which is linked to the inversion of the zero element. In their work, the authors state that conditional branching for the zero element can be avoided by changing two table elements: \(\log[0] = 2^n - 1\) and \(\text{alog}[2^n - 1] = 0\). However, because an inversion is defined as

\[
\alpha^{-1} = \text{alog}[(2^n - 1) - \log[\alpha]]
\]

the inversion of zero will result in \(0^{-1} = \text{alog}[0] = 1 \neq 0\) and, moreover, the inversion of 1 will result in \(1^{-1} = \text{alog}[(2^n - 1) - \log[1]] = \text{alog}[2^n - 1] = 0 \neq 1\). As a matter of fact, by setting \(\log[0] = 0\), \(\log[1] = 2^n - 1\) and \(\text{alog}[2^n - 1] = 0\), we found a possibility to correct the \(\log\) and \(\text{alog}\) tables in such a way that both inversions will work properly, again. In their paper, a multiplication of two elements is defined as

\[
\alpha \cdot \beta = \text{alog}[^{\log[\alpha]}] + ^{\log[\beta]} \mod 2^n - 1
\]

However, when using this method a multiplication with zero will only always result in zero, if conditional branching is used. Based on the inversion and multiplication with the \(\log\) and \(\text{alog}\) tables the authors propose two different masking schemes which are supposed to provide a secure inversion. We have carefully implemented and tested both schemes. We observed that in both schemes there occur special cases when the s-box input, the mask or masked, intermediate variables are equal to zero and which will result in a faulty behavior of the proposed masking schemes. We believe that a correction of their approach is only possible with the use of conditional branches, which makes it susceptible to power-analysis attacks.

The countermeasures presented in [BGK04] and [OMPR05] are based on a similar idea. In both papers, the authors assume that the inversion operation is computed step-by-step, either as exponentiation or with composite field arithmetic. The exponentiation method is advertised for software implementations and described in [BGK04]. The composite-field method is advertised for hardware implementations and is described in detail in [OMPR05]. Both methods do not seem to be particularly suited for 8-bit software implementations. However, as we will show in Section 4.3, especially the composite-field method can be adapted in such a way that it is suitable for 8-bit platforms.

\(^1\)i.e. for a chosen generator
4.2. AES AVR Smart Card Implementation

In this section, we present a reference software implementation of the AES for the AVR architecture. The target platform is an Atmel ATM163 RISC microcontroller embedded within a smart card [Atmb]. In order to achieve maximum performance and due to the limited hardware resources of ATM163 microcontroller, the AES was programmed in assembly. The smart card is running the *Simple Operating System for Smartcard Education (SOSSE)* [Mat], which is an open source operating system and conforms to the widely accepted smart card standard ISO 7816 [Int]. After programming the AES in a simulation environment and successfully validated with test vectors, its source code must be linked with the operating system SOSSE. The compilation of SOSSE then results in a single binary file, which has to be uploaded into the Flash ROM of the ATM163 microcontroller.

4.2.1. Atmel ATM163 microcontroller

As a target platform, smart cards with an Atmel ATM163 RISC microcontroller were chosen, since these smart cards are inexpensive (unit price about $14.00\textsuperscript{2}2), very flexible and freely available\textsuperscript{3}. Moreover, ATM163 smart cards can be purchased without signature of a Non Disclosure Agreement (NDA) with Atmel, since the ATM163 \(\mu\)C microcontroller does not provide any additional cryptographic or security related functionality. Also, the development environment *AVR Studio 4.0* is provided by Atmel [Atma] free of charge.

The ATM163 microcontroller is based on the 8-bit Atmel AVR RISC architecture [Atmb] which is a *Harvard* architecture, i.e., program memory and data memory are strictly separated. The ATM163 \(\mu\)C provides 16 KB of internal Flash ROM used as program memory, 1024 bytes of internal SRAM used as volatile data memory and 512 bytes of internal EEPROM used as non-volatile data memory. The smart card also contains an additional 256 KB EEPROM chip which is wired to the ATM163. This external EEPROM chip can also be used as non-volatile data memory. The wiring of the ATM163 \(\mu\)C with the external EEPROM and with the smart card contact pads is shown in Figure 4.1. Pads C1 and C5 are connected to the supply voltage (5 V and ground), pad C2 is used to reset the ATM163 \(\mu\)C, pad C3 provides the chip with a clock signal (typically 3.57\(\text{MHz}\)) and pad C7 is used as a bidirectional I/O transmission line\textsuperscript{4}.

The RISC architecture of the ATM163 features 32 internal 8-bit registers which are directly connected with the *Arithmetic Logic Unit (ALU)* of the processor. This allows

\textsuperscript{2}This price dates from March 2003.
\textsuperscript{3}These smart cards are often used illegally to clone original cards used in pay TV systems.
\textsuperscript{4}Pads C4 and C8 are used to program the Flash ROM of the ATM163 and are not conform with ISO7816.
Figure 4.1.: Wiring of the ATM163 and the external EEPROM with the smart card contact pads.

Simultaneous access of two registers by the ALU within a single clock cycle. Most instructions are executed within one clock cycle which results in a throughput of nearly one instruction per clock cycle [Atmb]. Smart cards are typically clocked externally by the card reader with a frequency of 3.57 MHz, thus, the smart card used in this project will achieve a performance close to 3.57 MIPS. The most important features of the ATM163 are:

- 130 RISC instructions, most instructions are executed within a single cycle,
- a maximum clock frequency of 8 MHz,
- 32 internal 8-bit registers directly connected with the ALU,
- 1024 Bytes of internal SRAM used as volatile data memory,
- 512 bytes of internal EEPROM used as non-volatile data memory,
- and 16 KB of internal Flash ROM used as program memory.

4.2.2. Simple Operating System for Smartcard Education (SOSSE)

The smart card operating system SOSSE was developed as an open source project under the Gnu Public License (GPL) [Mat]. Except for the communication routines, SOSSE is mainly programmed in Ansi C. SOSSE conforms to the widely accepted smart card
standard ISO 7816 [Int] and supports the T=0 communication protocol between card reader and smart card. The T=0 protocol was standardized in 1989 in the norm ISO 7816-3 and it is the most widely used communication protocol in smart card applications. For example, it is used by smart cards in *Global System for Mobile communications (GSM)* mobile phones [RE02]. The T=0 protocol is byte oriented, asynchronous and half-duplex. The card reader always functions as a master and the smart card as a slave, i.e., the card reader first sends a so-called *Application Protocol Data Unit (APDU)* command to the smart card, the smart card microcontroller executes the corresponding function and sends a response back to the card reader.

The transmission length of a bit sent over the serial I/O line is derived from the clock frequency with the fixed divisor 372. At a clock frequency of 3.57 MHz this results in a bit rate of 9600 bps. In the T=0 protocol, each byte is transmitted together with a 4-bit overhead (start, stop and parity bits), therefore, the actual data rate is 6400 bps which corresponds to a transmission time of 1.25 ms per data byte.

### 4.2.3. Properties of the AES

The AES is a symmetric key block cipher [DR02]. The block length is 128 bits and the key length can either be 128, 192 or 256 bits. However, in this project only key lengths of 128 bits are considered in order to restrict the required smart card resources. Inside AES, the plain-/ciphertext is stored within a $(4 \times 4)$ byte state matrix. During encryption the following transformations are executed in ten consecutive rounds:

- on-the-fly key schedule algorithm which recursively deduces the current 128-bit round key from the previous round key,

- byte-wise exclusive-or addition of the current 128-bit round key with the elements of the state matrix,

- shift row transformation which cyclically left shifts the rows of the state matrix,

- S-box transformation which is byte-wise, bijective and non-linear substitution of each element of the state matrix,

- mix column transformation which diffuses all four bytes of a column of the state matrix using a linear transformation in $GF(2^{32})$.

Since all AES transformations are bijective, a decryption equals an encryption with inverted transformations and round keys in reverse order.
4.2.4. Reference Implementation of the AES

In order to compare various masked variants of the AES, the plain AES was first programmed in assembly and implemented in the ATM163 smart card without any masking countermeasures. This implementation served as a reference and was used to simplify the debugging of further AES variants. The details of this implementation are given in Table 4.2.4. It encrypts a 128-bit plaintext within 4419 clock cycles. At a clock frequency of 3.57 MHz this corresponds to a duration of 1.23 ms which is less than the transmission length of one byte. As resources, the implementation requires 1288 bytes of Flash ROM and 34 bytes of SRAM.

Using an oscilloscope, we measured that the overall duration of an AES encryption including the transmission of the 16-byte plaintext to the card and receiving the corresponding 16-byte ciphertext from the card is about 60 ms. Since the transmission length of a byte via the I/O line is 1.25 ms, it becomes clear that the major bottleneck in the scenario is the slow data rate between card reader and smart card. Therefore, AES implementations as well as other block or stream ciphers running on smart cards are not suited to process continuous data streams.

4.3. First Order Masking of AES

4.3.1. Composite Field Based Inversion

The only difficult part in masking AES is to mask the S-box operation. The S-box operation is composed of two parts: an inversion in $GF(2^4)$ and an affine mapping. Again, masking the affine part is easy, so we focus on the non-linear inversion operation only. Our goal is that all input and output values in the computation of the inverse are masked. According to [OMPR05], a masked input can be transformed to the composite field $GF(2^4) \times GF(2^4)$ with an isomorphic mapping, where it can be securely and efficiently inverted, and finally transformed back to the $GF(2^8)$. The inversion operation
in the composite field can be computed as follows:

\[
((a_h + m_h)x + (a_l + m_l))^{-1} = (a_h' + m_h')x + (a_l' + m_l')
\]

\[
a_h' + m_h' = f_a((a_h + m_h), (d' + m_d'), m_h, m_h', m_d)
\]

\[
a_l' + m_l' = f_a((a_h' + m_h'), (a_l + m_l), (d' + m_d'), m_l, m_h', m_l', m_d')
\]

\[
d + m_d = f_d((a_h + m_h), (a_l + m_l), p_0, m_h, m_l, m_d)
\]

\[
d' + m_d' = f_d(d + m_d, m_d, m_d')
\]

The functions \(f_a, f_{a'}, f_d\) and \(f_{d'}\) are functions on \(GF(2^4)\). This calculation of a masked inversion operation is based on the composite field approach that is described in detail in [WOL02]. Whereas in [OMPR05] this approach is applied to hardware implementations and has been extended to work in so-called tower fields, we pursue a different approach. We show that these formulae can be mapped to a sequence of table look-ups and XOR operations. We show how to define tables which only require little space in memory. Furthermore, we show that only a small number of table look-ups are required to calculate the formulae.

**Pre-computed Tables**

We compute a number of tables that do the operations in \(GF(2^4)\) and store them in memory:

\[
T_{d_1} : ((x + m), m) \mapsto x^2 \times p_0 + m
\]

\[
T_{d_2} : ((x + m), (y + m')) \mapsto ((x + m) + (y + m')) \times (y + m')
\]

\[
T_n : ((x + m), (y + m')) \mapsto (x + m) \times (y + m')
\]

\[
T_{inv} : ((x + m), m) \mapsto x^{-1} + m.
\]

All tables (or functions) take two elements of \(GF(2^4)\) as inputs and give an element of \(GF(2^4)\) as output. With these 4 Tables, we can compute formulae (4.2)-(4.5). In order to map \(GF(2^8)\) elements to \(GF(2^4) \times GF(2^4)\) elements and vice versa, we need two more tables \(Map : x \mapsto z\) and \(Map^{-1} : z \mapsto x\). \(Map\) takes an element \(x\) of \(GF(2^8)\) as input and gives an element \(z\) of \(GF(2^4) \times GF(2^4)\) as output. \(Map^{-1}\) works vice versa. We assume that for all tables the input masks and the output masks are identical. Hence, the size of one table is at most 256 bytes and so we can pre-compute all tables and store them in read-only memory (ROM), since there is no need to compute them during
run-time. This is a significant advantage over the use of $MS - box()$ tables. They have to be computed for every new mask $m$ during run-time or at least at the invocation of a new AES encryption run.

**Masked Inversion**

First, we have to compute the masked value of $d$, i.e., $d + m_d = d + m_h$ according to Equation 4.4:

$$f_d(a_h + m_h, a_l + m_l, m_h, m_l, m_h) = T_{d_1}(a_h + m_h, m_h) + T_{d_2}((a_h + m_h), (a_l + m_l)) + T_m((a_h + m_h), m_l) + T_m((a_l + m_l), m_l) + T_m((m_h + m_l) m_l)$$

(4.6)

It is easy to check that the result will be indeed $a_h^2 \times p_0 + a_h \times a_l + a_l^2 + m_h$. For this computation we need five table look-up operations (TLs), four XOR operations and an additional XOR operation to compute $(m_h + m_l)$ which is used as input in $T_m((m_h + m_l), m_l)$. Note that the results of $T_m((a_h + m_h), m_l)$ and $T_m((a_l + m_l), m_l)$ are used again in equations (4.8) and (4.9), respectively, therefore it is a good idea to store these results and reuse them later on in order to save these two look-up operations.

In the next step we compute the inverse of the masked $d$ with one more table look-up operation:

$$f_{d'}(d + m_h, m_h, m_h) = T_{inv}(d + m_h, m_h).$$

(4.7)

In order to derive $f_{a_h}()$, we first compute $d^{-1} + m_l$ by one XOR addition with the term $(m_h + m_l)$. Then $f_{a_h}(a_h + m_h, d^{-1} + m_l, m_h, m_h, m_l)$ can be computed as follows:

$$f_{a_h}(a_h + m_h, d^{-1} + m_l, m_h, m_l, m_l) = T_m(a_h + m_h, d^{-1} + m_l) + m_h + T_m(d^{-1} + m_l, m_l) + T_m(a_h + m_h, m_l) + T_m(m_h, m_l).$$

(4.8)

This computation gives as output $a_h \times d^{-1} + m_h$. For this computation, we need three new table look-up operations and four XOR operations in total.

In the last step we derive $f_{a_i}(a_h \times d^{-1} + m_h, a_l + m_l, d^{-1} + m_l, m_h, m_l, m_l)$. Hence, we calculate:

$$f_{a_i}(a_h \times d^{-1} + m_h, a_l + m_l, d^{-1} + m_l, m_h, m_l, m_l) = T_m((a_l + m_l), (d^{-1} + m_h) + m_l) + T_m(d^{-1} + m_h, m_l) + T_m(a_l + m_l, m_l) + f_{a_h} + m_h + T_m(m_l, m_l).$$

(4.9)
This gives \( a_i \times d^{-1} + a_h \times d^{-1} + m_i \) as a result. Note that the term \( T_m(m_h, m_i) \) occurs in the computation of \( f_{ah} \) and \( f_{ai} \). Hence, by also storing \( f_{ah} + T_m(m_h, m_i) \) during the computation of \( f_{ah} \) and using this term during the computation of \( f_{ai} \), one additional table look-up and one XOR can be saved. Therefore, for this computation we only need two additional table look-ups and five XOR operations.

Prior to the inversion in \( GF(2^4) \times GF(2^4) \) we need to map the 8-bit values (elements in \( GF(2^8) \)) to \( 2 \times 4 \)-bit values (elements in \( GF(2^4) \times GF(2^4) \)). This is done by a table look-up as well. Mapping back from \( GF(2^4) \times GF(2^4) \) to \( GF(2^8) \) can be achieved with an additional look-up table. Moreover, it makes sense to combine the isomorphic mapping from \( GF(2^4) \times GF(2^4) \) to \( GF(2^8) \) with the affine transformation that is part of S-box and use only one table for both.

**Total costs of a Masked Inversion**

If we review the number of table look-ups (TLs) and XOR additions required for an entire masked AES S-box operation, we need five TL operations and four XOR additions in equation 4.6, one TL operation in Equation 4.7, three TL operations and four XOR additions in Equation 4.8, two TL operations and five XOR additions in Equation 4.9. Furthermore, we need three TL operations for the isomorphic transformations: two TL operations to map the masked inversion input and the mask to \( GF(2^4) \times GF(2^4) \) and one TL operation to map the masked result of the inversion back to \( GF(2^8) \) and perform the affine transform. This sums up to a total of 14 table look-up operations and 15 XOR operations.

**Theoretical Security Analysis**

In this section we show that all data-dependent intermediate masked values that are computed during the masked inversion operation are statistically independent from the unmasked values. Hence, we follow the definition of security that was introduced in [CJR99b] and strengthened in [BGK04]. The values that we have to investigate are the outputs of the functions (tables) \( T_{ah}, T_m, T_{inv}, Map, Map^{-1} \) and all intermediate values that occur after an XOR operation. In [OMPR05] it has been shown in Lemma 5 that a sum of independent masked values will again be independent from the unmasked values as long as an independent mask is used during the summation. Furthermore, in Lemmas 1–4 it has been shown that the XOR operation, as well as the masked multiplication and the masked squaring are secure in the sense that their output is statistically independent from the plaintext input.

**Lemma 1** Let \( x \in GF(2^n) \) be arbitrary and let \( p_0 \in GF(2^n) \) be an arbitrary but fixed value. Let \( m \in GF(2^n) \) be independently and uniformly distributed in \( GF(2^n) \). Then
\[ T_{d_1}(x + m, m) = x^2 \times p_0 + m \text{ is uniformly distributed regardless of } x. \text{ Therefore, the distribution of } x^2 \times p_0 + m \text{ is independent of } x. \]

**Proof:** As \( x \) is an element of the binary extension field, the element \( x^2 = (\sum_i a_i \alpha^i)^2 = \sum_i a_i \alpha^{2i} \) with \( a_i \in \{0, 1\} \) is in \( GF(2^n) \) as well. Hence, all elements of \( GF(2^n) \) are quadratic residues and thus \( x^2 \) is uniformly distributed on \( GF(2^n) \). Consequently, also \( x^2 \times p_0 \) and \( x^2 \times p_0 + m \) are uniformly distributed.

For the independency of the output of \( T_{d_2} \) we reuse Lemma 2 of [BGK04].

**Lemma 2** Let \( x, y \in GF(2^n) \) be arbitrary. Let \( m, m' \in GF(2^n) \) be independently and uniformly distributed in \( GF(2^n) \). Then the probability distribution of \( T_{m}(x+m, y+m') = (x+m) \times (y+m') \) is

\[
p((x+m) \times (y+m') = i) = \begin{cases} \frac{2^{n+1} - 1}{2^{2n}} & , \text{if } i = 0, \text{ i.e., if } m = x \text{ or } m' = y \\ \frac{2^{n+1} - 2^{n+1} + 1}{2^{2n}} & , \text{if } i \neq 0. \end{cases}
\]

Therefore, the distribution of \( (x+m) \times (y+m') \) is independent of \( x \) and \( y \).

Lemma 3 follows directly from Lemma 2 and the observation that all elements of \( GF(2^n) \) are quadratic residues.

**Lemma 3** Let \( x, y \in GF(2^n) \) be arbitrary. Let \( m, m' \in GF(2^n) \) be independently and uniformly distributed in \( GF(2^n) \). Then the probability distribution of \( T_{d_2}(x+m, y+m') = (x+m) \times (y+m') + (y+m')^2 \) is

\[
p((x+m) \times (y+m') + (y+m')^2 = i) = \begin{cases} \frac{2^{n+1} - 1}{2^{2n}} & , \text{if } i = 0, \text{ i.e., if } m = x \text{ or } m' = y \\ \frac{2^{n+1} - 2^{n+1} + 1}{2^{2n}} & , \text{if } i \neq 0. \end{cases}
\]

Therefore, the distribution of \( (x+m) \times (y+m') \) is independent of \( x \) and \( y \).

The independence of \( T_{inv}(x + m, m) = x^{-1} + m \) is clear as the inversion operation is bijective (note that the zero element is mapped to the zero element) and the XOR of any \( a + m \) is independent from \( a \). The mappings between \( GF(2^8) \) and \( GF(2^4) \times GF(2^4) \) are bijections and therefore their masked output is independent from the unmasked input in a statistical sense.

Based on these results we may conclude that the algorithm for computing masked inversion complies to the definition of security used in [BGK04].
4.3.2. Implementation of the Composite Field Masking Scheme

In our following analysis we regard the implementation of the S-box transformation in assembly on a smart card based on the 8-bit Atmel AVR architecture. In total, we require six pre-computed tables which can be stored in read-only memory (ROM). Table $T_{d_1}$ takes two $GF(16)$ elements as input and gives one $GF(16)$ element as output. The same holds for $T_{d_2}$, $T_m$ and $T_{inv}$, as well. Hence, these four tables map an 8-bit input to a 4-bit output value.

In a software implementation there are two possibilities how the tables $T_{d_1}, T_{d_2}, T_m$ and $T_{inv}$ can be stored in memory on an 8-bit architecture. In a compact representation, each byte of these four tables stores two 4-bit output values, hence, each table requires 128 bytes in ROM and the four tables altogether require $4 \times 128 = 512$ bytes in ROM. The disadvantage of this compact representation is based on the fact that a few instructions are required after each table look-up to either erase the unwanted upper 4-bit half or to shift the upper 4-bit half by four bits to the right in order to erase the unwanted lower 4-bit half. These instructions are not required, if each byte of the tables $T_{d_1}, T_{d_2}, T_m$ and $T_{inv}$ only stores a single 4-bit result and the upper 4-bit half is always set to zero. This representation is more efficient in terms of clock cycles, but requires $4 \times 256 = 1024$ bytes in ROM. In the following we will only regard the efficient representation. The two isomorphic mappings from $GF(2^8)$ to $GF(2^4) \times GF(2^4)$ and back from $GF(2^4) \times GF(2^4)$ to $GF(2^8)$ deliver a $GF(2^4) \times GF(2^4)$ and a $GF(2^8)$ element as output, i.e. these two tables map an 8-bit input to an 8-bit output. Hence, in total we need $4 \times 256 + 2 \times 256 = 1536$ bytes to store all six tables in ROM.

The AVR architecture is a RISC design and thus provides 32 internal registers. A TL operation which reads an 8-bit value from a table stored in ROM to an internal register takes five clock cycles. A TL operation which reads an 8-bit value from a table stored in RAM to an internal register or writes an 8-bit value to a table stored in RAM takes four clock cycles. The XOR addition of two internal registers requires only a single clock cycle.

In an unmasked AES software implementation every S-box step would only require a single TL operation. If a standard masked table look-up, such as described in Section 4.1 is used, the S-box table would be stored in ROM and the masked tables would be derived from it prior to an AES encryption/decryption and then stored in RAM. If only one encryption is performed, this pre-computation would very likely be done for the 16 masks, only, and thus require $16 \times 256 = 4096$ bytes in RAM. During the encryption/decryption of AES only a single TL operation would be required for each S-box step. However, the pre-computation of the each masked table in RAM would require 256 XOR additions to mask the table index, 256 TL operations to read the unmasked table entries from ROM, 256 XOR additions to mask the table entries and finally 256 TL operations to store the masked table in RAM. If tables are generated in such a way for 16 different masks, this will result in pre-computational costs of $16 \times (256 + 256 \times 5 + 256 + 256 \times 4) = 45056$
clock cycles. If several encryption operations would be performed after each other and the same set of masks is used over and over again, the pre-computational costs occur only once. However, from a security point of view it is advisable to update the masks as often as possible. Another possibility is to store all masked tables in ROM. However, this would require 256 × 256 = 64 KB in ROM which might exceed the limitations in constrained environments such as smart cards. As stated in section 4.3.1, when using our proposal an entire S-box step for an arbitrary mask requires 14 TL operations and 15 XOR additions which results in 14 × 5 + 15 = 85 clock cycles. For an entire AES encryption this results in 10 × 16 × 85 = 13600 clock cycles. Our method requires 1536 bytes in ROM and no RAM, moreover, no pre-computation needs to be performed. In Table 4.3.2 the costs of various masked and unmasked AES implementations are compared. Our proposal is referred to as "New" in Table 4.3.2.

<table>
<thead>
<tr>
<th></th>
<th>ROM</th>
<th>RAM</th>
<th>PRE-TL</th>
<th>PRE-XOR</th>
<th>TL</th>
<th>XOR</th>
<th>cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>unmasked</td>
<td>256</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>160</td>
<td>0</td>
<td>800</td>
</tr>
<tr>
<td>256 fixed masks</td>
<td>64 KB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>160</td>
<td>0</td>
<td>800</td>
</tr>
<tr>
<td>single mask</td>
<td>256</td>
<td>256</td>
<td>512</td>
<td>512</td>
<td>160</td>
<td>0</td>
<td>3456</td>
</tr>
<tr>
<td>16 masks</td>
<td>256</td>
<td>4096</td>
<td>8192</td>
<td>8192</td>
<td>160</td>
<td>0</td>
<td>45696</td>
</tr>
<tr>
<td>proposed S-box</td>
<td>1536</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2240</td>
<td>2400</td>
<td>13600</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of various AES software implementations with regard to code size and speed for a single encryption.

Hence, the complexity of our proposal is lower in terms of memory and operations for a single encryption. If only a single mask is used, our proposal is about four times slower for a single encryption, however, our approach does not require any RAM. Furthermore, it has been pointed out in [GT02] that the usage of a single mask in AES may allow simple second-order DPA attacks, which can be avoided by the usage of 16 different masks in each round. If encryptions are repeated several times with the same set of 16 masks, our proposal will be slower after four encryptions, but will always require less memory.

4.3.3. Power Analysis of the new Scheme

In order to confirm the security claims that we made in Section 4.3.1 and to assess the practical security of our implementation, we performed DPA attacks on an AES implementation based on our new inversion scheme. The target hardware was a smart card based on the AVR architecture. DPA attacks were performed in two independent experiments. The first time we performed DPA attacks on the implementation with the masking countermeasure switched off, i.e. all masks were fixed to zero. The second time we performed DPA attacks on the implementation with the masking countermeasure
being active, i.e. all masks were randomly generated. In both experiments 1000 random plaintexts were encrypted and the corresponding power traces were measured using a digital oscilloscope with a sampling rate of 100 MSa/s and a current probe. The resulting differential traces are shown in Figure 4.2 and Figure 4.3.

![Figure 4.2: DPA of the AES with no active countermeasure.](image1)

![Figure 4.3: DPA of the AES with our new masked s-box scheme.](image2)

It is obvious that the DPA of the unprotected AES implementation is successful, since a distinct correlation peak is contained in plot 4.2 for the correct key hypothesis. However, as shown in plot 4.3 the DPA of our new protected AES scheme was not succesful.

### 4.4. Higher Order Masking of AES

The development of masking schemes which aim at securing implementations of cryptographic algorithms against side channel attacks is a topic of ongoing research. Unfortunately, to our knowledge most of these countermeasures address first-order side channel attacks, only. In Section 4.4.1, we discuss the theoretical background of higher order DPA. Subsequently, in Sections 4.4.2 to 4.4.4 we present the expected measurement costs an adversary has to face for various power leakage models and we show that the number
of measurements increases exponentially with the order of the DPA attack. Finally, in Sections 4.4.5 to 4.4.7, we propose a scheme based on multiple masks which protects AES implementations against higher order DPA attacks. We have implemented various variants of this masking scheme in assembly for the AVR architecture and present performance details in Section 4.4.8.

4.4.1. HODPA: Theoretical Issues

We assume an adversary encrypts $N$ plaintexts $X_j$ and measures the corresponding power traces $P_j(t)$. As discussed in [CJR+99b, Mes00b, AG03, WW04], we define a DPA of order $d$ as the correlation of the product of $d$ power signals $P_j(t_1),...,P_j(t_d)$ with a selected function $f$ of the known plaintext $X_j$ and a key hypothesis $K_h$.

$$\rho(\prod_{i=1}^{d} P(t_i), f(X_j, K_h)) = \frac{COV[\prod_{i=1}^{d} P(t_i), f(X_j, K_h)]}{\sqrt{V[\prod_{i=1}^{d} P(t_i)]} \sqrt{V[f(X_j, K_h)]}}$$

(4.10)

Since the adversary is generally only able to measure a finite number $N$ of power traces $P_j(t)$, the correlation coefficient $\rho = \lim_{N \to \infty} \hat{\rho}(N)$ is estimated using approximated covariance and variances.

$$COV[\prod_{i=1}^{d} P(t_i), f(X_j, K_h)] = \frac{1}{N} \sum_{j=0}^{N-1} \prod_{i=1}^{d} P_j(t_i) f(X_j, K_h)$$

$$- \left( \frac{1}{N} \sum_{j=0}^{N-1} \prod_{i=1}^{d} P_j(t_i) \right) \left( \frac{1}{N} \sum_{j=0}^{N-1} f(X_j, K_h) \right)$$

(4.11)

$$V[\prod_{i=1}^{d} P(t_i)] = \frac{1}{N} \sum_{j=0}^{N-1} \left( \prod_{i=1}^{d} P_j(t_i) - \frac{1}{N} \sum_{j=0}^{N-1} \prod_{i=1}^{d} P_j(t_i) \right)^2$$

(4.12)

$$V[f(X, K_h)] = \frac{1}{N} \sum_{j=0}^{N-1} \left( f(X_j, K_h) - \frac{1}{N} \sum_{j=0}^{N-1} f(X_j, K_h) \right)^2$$

(4.13)

4.4.2. Multi-Bit SODPA of the AES S-box Input

We suppose an adversary performs a second-order DPA based on an $l$-bit key hypothesis $K_h$ against the masked input of an AES S-box in round one with $1 \leq l \leq n = 8$. Furthermore, we assume that a random mask $M$ leaks at time $t_1$, the masked S-box input $X \oplus K \oplus M$ leaks at time $t_2$ and that the adversary is able to measure the
corresponding power signals \( P(t_1) \) and \( P(t_2) \). If the power contribution \( \epsilon \) of all bits is equal\(^5\) and coupling effects among the bits are neglected, the power signals \( P(t_1) \) and \( P(t_2) \) can be modelled as

\[
P(t_1) = \epsilon \cdot \sum_{i=0}^{n-1} M[i] + N_1 \quad \text{and} \quad P(t_2) = \epsilon \cdot \sum_{i=0}^{n-1} (X \oplus K \oplus M)[i] + N_2
\]

where \( N_1, N_2 \sim N(0, \sigma^2) \) denote additive Gaussian noise terms and \( n \) denotes the bit length of all intermediate variables, i.e. in the case of AES \( n = 8 \). Then, the covariance of the product \( P(t_1) \cdot P(t_2) \) and the Hamming weight of the hypothesized S-box input \( W(X \oplus K_h) \) is

\[
COV[P(t_1) \cdot P(t_2), W(X \oplus K_h)] = COV[P(t_1) \cdot P(t_2), W(X \oplus K_h) - \frac{l}{2}] = E[P(t_1)P(t_2) \cdot (W(X \oplus K_h) - \frac{l}{2})] - E[P(t_1)P(t_2)] \cdot E[(W(X \oplus K_h) - \frac{l}{2})]
\]

\[
= E[\epsilon^2 \sum_{i=0}^{n-1} \sum_{h=0, h \neq i}^{n-1} \sum_{j=0}^{l-1} M[i](M \oplus X \oplus K)[h](X \oplus K_h)[j]] - \frac{l}{2} E[\epsilon^2 \sum_{i=0}^{n-1} M[i](M \oplus X \oplus K)[i]]
\]

\[
- \frac{1}{2} E[\epsilon^2 \sum_{i=0}^{n-1} \sum_{h=0, h \neq i}^{n-1} M[i](M \oplus X \oplus K)[h]] + \frac{1}{2} \epsilon^2 n(n-1) \cdot \frac{1}{4}
\]

\[
+ E[\epsilon^2 \sum_{i=0}^{n-1} \sum_{j=0}^{l-1} M[i](M \oplus X \oplus K)[i](X \oplus K_h)[j]] - \frac{l}{2} E[\epsilon^2 \sum_{i=0}^{n-1} M[i](M \oplus X \oplus K)[i]]
\]

\[
- \frac{1}{2} E[\epsilon^2 \sum_{i=0}^{n-1} M[i](M \oplus X \oplus K)[i]] + \frac{1}{2} \epsilon^2 n(n-1) \cdot \frac{1}{4}
\]

\[
+ E[N_2 \cdot \epsilon \cdot \sum_{i=0}^{n-1} M[i] \cdot \left( \sum_{i=0}^{l-1} X \oplus K_h[i] - \frac{l}{2} \right)]
\]

\[
+ E[N_1 \cdot \epsilon \cdot \sum_{i=0}^{n-1} (M \oplus X \oplus K)[i] \cdot \left( \sum_{i=0}^{l-1} X \oplus K_h[i] - \frac{l}{2} \right)]
\]

\[
+ E[N_1 \cdot N_2 \cdot \left( \sum_{i=0}^{l-1} X \oplus K_h[i] - \frac{l}{2} \right)] = \frac{1}{4} \cdot \epsilon^2 \cdot (\frac{1}{2} - u)
\]

\(^5\)Hamming weight model
where $u$ denotes the number of correctly guessed key bits, $0 \leq u \leq l$. The variance of the Hamming weight of the hypothesized S-box input is

$$V[W(X \oplus K_h)] = V[W(X \oplus K_h) - \frac{l}{2}] = \frac{l}{4}$$

The variance of the product $P(t_1) \cdot P(t_2)$ can be expressed as

$$V[P(t_1) \cdot P(t_2)] = V[\epsilon^2 \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} M[i](M \oplus X \oplus K)[j]]$$

$$\quad + V[N_2 \cdot \epsilon \cdot \sum_{i=0}^{n-1} M[i]]$$

$$\quad + V[N_1 \cdot \epsilon \cdot \sum_{i=0}^{n-1} M \oplus X \oplus K[i]]$$

$$\quad + V[N_1 \cdot N_2]$$

$$= \epsilon^4 \cdot n^2 \cdot \frac{1}{16} \cdot (3 + 2(n - 1)) + \epsilon^2 \cdot \sigma^2 [(1 + n)\sigma^2] + \sigma^4$$

This results in the correlation coefficient

$$\rho(P(t_1) \cdot P(t_2), W(X \oplus K_h)) = \frac{COV[P(t_1) \cdot P(t_2), W(X \oplus K_h)]}{\sqrt{V[P(t_1) \cdot P(t_2)]} \cdot \sqrt{V[W(X \oplus K_h)]}}$$

$$= \frac{COV[P(t_1) \cdot P(t_2), W(X \oplus K_h) - \frac{l}{2}]}{\sqrt{V[P(t_1) \cdot P(t_2)]} \cdot \sqrt{V[W(X \oplus K_h) - \frac{l}{2}]}}$$

$$= \frac{\frac{1}{4} \epsilon^2 (\frac{l}{2} - u)}{\sqrt{\frac{1}{16} \epsilon^4 \cdot n^2 (3 + 2(n - 1)) + \epsilon^2 \cdot \sigma^2 (1 + n) + \sigma^4}} \sqrt{\frac{l}{4}}$$

(4.14)

where $W(X \oplus K_h) \leq l$ denotes the Hamming weight of the guessed lower $l$ bits of the unmasked S-box input and $u$ denotes the number of correctly guessed key bits, $0 \leq u \leq l$. The expression can be simplified, if we assume that the power signals only depend on the Hamming weights ($\epsilon = 1$) and that the uncorrelated noise terms $N_1$ and $N_2$ are neglected ($\sigma = 0$).

$$\rho(P(t_1) \cdot P(t_2), W(X \oplus K_h)) = \frac{\frac{1}{4} (\frac{l}{2} - u)}{\sqrt{\frac{1}{16} \cdot n^2 (3 + 2(n - 1))}} \sqrt{\frac{l}{4}}$$

(4.15)

The AES S-box input in the first round is a linear function of a plaintext byte $X$ and a key byte $K$. As a result, the resulting correlation coefficient shows a linear characteristic.
4.4 Higher Order Masking of AES

It is proportional to the number of correctly guessed key bits and reaches its minimum (maximum)\(^6\), if all key bits are guessed correctly (incorrectly). Moreover, wrong key guesses which are close to the correct key guess (e.g. \(l - 1\) bits are guessed correctly and only one bit incorrectly) will result in a correlation coefficient which is close to its minimum. Therefore, DPA attacks usually focus on the output of non-linear functions, such as the AES S-boxes, because wrong key guesses will result in correlation coefficients which are clearly distinguishable from the correct key guess. Please note, however, that in [BCO04] it was shown that S-boxes which are not perfectly non-linear (e.g. the DES S-boxes) may result in ghost peaks.

4.4.3. Multi-Bit HODPA of the AES S-box Output

We suppose an adversary performs a DPA of order \(d\) against the S-box output, i.e., the adversary correlates the product of \(d\) power signals with a selected function of the unmasked S-box output based on the key hypothesis \(K_h\). Moreover, we assume that the leakage of a variable is equal to its Hamming weight (i.e. \(\epsilon = 1\) and \(\sigma = 0\)) and that the adversary knows that the Hamming weights of \(d - 1\) random masks \(M_1, \ldots, M_{(d-1)}\) leak at times \(t_1, \ldots, t_{(d-1)}\) and that the masked S-box output \(S(X \oplus K) \oplus M_1 \oplus \ldots \oplus M_{(d-1)}\) leaks at time \(t_d\). Hence, given are \(d\) power signals \(P(t_i)\) according to the noise-free Hamming weight model

\[
P(t_1) = W(M_1) \\
... \\
P(t_{(d-1)}) = W(M_{(d-1)}) \\
P(t_d) = W(S(X \oplus K) \oplus M)
\]

with \(M = M_1 \oplus \ldots \oplus M_{(d-1)}\). Let \(n\) be the bit length of all intermediate variables, i.e. in the case of AES \(n = 8\). Then, the covariance of the product \(\prod_{i=1}^{d} P(t_i)\) and the Hamming weight of the hypothesized S-box output \(W(S(X \oplus K_h))\) is

\[
COV[\prod_{i=1}^{d} P(t_i), W(S(X \oplus K_h))] = COV[\prod_{i=1}^{d} P(t_i), W(S(X \oplus K_h)) - \frac{n}{2}] \\
= E[\prod_{i=1}^{d} P(t_i) \cdot (W(S(X \oplus K_h)) - \frac{n}{2})] - E[\prod_{i=1}^{d} P(t_i)] \cdot E[(W(S(X \oplus K_h)) - \frac{n}{2})] \\
= E[\prod_{i=1}^{d} P(t_i) \cdot (W(S(X \oplus K_h)) - \frac{n}{2})] - \frac{n}{2} \cdot E[\prod_{i=1}^{d} P(t_i)] \\
= (\frac{n}{2})^d
\]

\(^6\)the minimum and maximum have an equal magnitude, i.e. the correlation coefficient is a symmetric function.
The variance of the Hamming weight of the hypothesized S-box output is
\[
V[W(S(X \oplus K_h))] = V[W(S(X \oplus K_h))] - \frac{n}{2} = \frac{n}{4}
\]
The variance of the product \( \prod_{i=1}^{d} P(t_i) \) is
\[
V[\prod_{i=1}^{d} P(t_i)] = E[\prod_{i=1}^{d} P^2(t_i)] - E[\prod_{i=1}^{d} P(t_i)]^2
\]
\[
= \sum_{i=1}^{d} \sum_{j=1}^{d} E[W^2(M_i) \cdots W^2(M_{(d-1)}) \cdot W^2(S(X \oplus K_h))] \\
\quad \quad \quad \quad \quad \quad \quad \quad \quad - E[W(M_i) \cdots W(M_{(d-1)}) \cdot W(S(X \oplus K_h))]^2
\]
\[
= \left( \frac{n}{4} + \frac{n^2}{4} \right)^d - \left( \frac{n}{2} \right)^{2d}
\]
For a correct key guess \( (K_h = K) \) this results in the correlation coefficient
\[
\rho(\prod_{i=1}^{d} P(t_i), W(X \oplus K_h)) = \frac{\text{COV}[\prod_{i=1}^{d} P(t_i), W(X \oplus K_h)]}{\sqrt{V[\prod_{i=1}^{d} P(t_i)]} \cdot \sqrt{V[W(X \oplus K_h)]}}
\]
\[
= \frac{\text{COV}[\prod_{i=1}^{d} P(t_i), W(X \oplus K_h) - \frac{n}{2}]}{\sqrt{V[\prod_{i=1}^{d} P(t_i)]} \cdot \sqrt{V[W(X \oplus K_h) - \frac{n}{2}]} }
\]
\[
= \frac{2^{-(d+1)} \cdot n \cdot (-1)^{(d+1)}}{\sqrt{\left( \left( \frac{n}{4} + \frac{n^2}{4} \right)^d - \frac{n^2}{4} \right)}} (\frac{n}{4})^{(d+1)}
\]
(4.16)
where \( n \) denotes the bit length of all intermediate variables. If the key is not guessed correctly, the correlation coefficient is approximately zero [BK02, BCO04]. In the case of AES, \( n = 8 \), i.e. 8 bits of \( K_h \) must be guessed to predict the S-box output. Thus, the correlation coefficient of the correct key guess reduces to
\[
\rho(\prod_{i=1}^{d} P(t_i), W(S(X \oplus K))) = \frac{2^{(2d-4)} \cdot (-1)^{(d+1)}}{\sqrt{(9d-8d) \cdot 2^{(d+1)}}} \quad \text{if } K_h = K
\]
(4.17)
while wrong key hypotheses, i.e. \( K_h \neq K \), result in correlation coefficients which converge to zero for an increasing number of measurements \( N \) due to the non-linear characteristics of the AES S-box. In Figures 2.4 and 2.4, the results of a simulated second-order
and third-order DPA based on the Hamming weight model are shown. In both cases the correlation coefficients corresponding to the correct key hypotheses are clearly visible, however, significantly more measurements (≈ a factor of $10^2$) are required to successfully perform the third-order DPA\(^7\).

Table 4.3: Correlation coefficients of a successful multi-bit DPA for a given order \(d\) predicting the AES S-box output. The leakage is presumed to be equal to the Hamming weight of intermediate variables (no additive noise).

<table>
<thead>
<tr>
<th>DPA Order (d)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corr. Coeff. (\rho)</td>
<td>1</td>
<td>-0.0857</td>
<td>0.0085</td>
<td>-8.901 \times 10^{-4}</td>
<td>9.638 \times 10^{-5}</td>
<td>-1.064 \times 10^{-5}</td>
</tr>
</tbody>
</table>

In Table 4.3, the correlation coefficients of correct key hypotheses for DPA attacks of orders \(d = 1, ..., 6\) are listed. Please note that the correlation coefficients approximately decrease by a factor of 10 with order \(d\) and, moreover, feature alternating signs.

In order to define some kind of quality rating regarding a HODPA attack, we need to define a signal-to-noise ratio (SNR) which expresses how much the estimated correlation coefficient of the correct key hypothesis deviates from the estimated correlation.

\(^7\) Also note that the magnitude of the correlation coefficients in the third-order plot has approx. decreased by a factor of 10.
<table>
<thead>
<tr>
<th>DPA Order $d$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>31.14</td>
<td>3941.67</td>
<td>415513.67</td>
<td>44383112.11</td>
</tr>
<tr>
<td>$\approx 3.13 \cdot 10^1$</td>
<td>$\approx 3.94 \cdot 10^3$</td>
<td>$\approx 4.16 \cdot 10^5$</td>
<td>$\approx 4.44 \cdot 10^7$</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.4: Number of measurements $N$ required to achieve an $|SNR| \geq 5$ in simulated multi-bit DPA attacks for a given order $d$ predicting the AES S-box output. (averaged over 100 simulated DPA attacks for each order $d$). The leakage is presumed to be equal to the Hamming weight of intermediate variables (no noise).

Experimental results showed that an $|SNR|$ of $\geq 5$ is a reasonable threshold, i.e. it results in satisfactory HODPA attacks for which the correct key guess is clearly distinguishable from wrong key guesses. Table 4.4 lists the average number of measurements $N$ required to achieve an $|SNR|$ of $\geq 5$. These numbers were derived from statistical simulations, i.e. for a given order $d$ 100 simulated HODPA attacks were performed. The numbers given in Table 4.4 clearly show that the measurement costs grow exponentially with DPA order $d$ (see [CJR+99b]).

Please note that in [Man04, SPRQ05, OMHT06] a method was proposed, which makes it possible to analytically assess the required number of measurements for a successful DPA attack based on Fisher’s $Z$-transformation and statistical confidence intervals. Let us assume an adversary performs several equal HODPA attacks, i.e. each with the same number of measurements $N$ and exactly the same measurement setup parameters, and estimates the correlation coefficient $\hat{\rho}(N)$ for each attack using a fixed key hypothesis. An analysis of the estimated correlation coefficients reveals that their mean approximates the actual correlation coefficient $\rho$, but that they are not normally distributed. However, Fisher $Z$-transformed correlation coefficients are known to have an approximate normal distribution, which eventually makes it possible to define statistical confidence intervals in order to distinguish correlation coefficients of false key hypotheses from the correlation coefficient of the correct key hypothesis.

$$z = \frac{1}{2} \cdot \ln \left( \frac{1 + \hat{\rho}(N)}{1 - \hat{\rho}(N)} \right)$$

$$\mu = \frac{1}{2} \cdot \ln \left( \frac{1 + \rho}{1 - \rho} \right)$$

$$\sigma^2 = \frac{1}{N - 3}$$
Hence, Z-transformed correlation coefficients are normally distributed with mean $\mu$ and variance $\sigma^2 = \frac{1}{N-3}$. Next, we can distinguish the mean of Z-transformed correlation coefficients for correct and incorrect key hypotheses:

$$\mu_{K=K_h} = \frac{1}{2} \cdot \ln\left(\frac{1 + \rho}{1 - \rho}\right)$$  
$$\mu_{K\neq K_h} = \frac{1}{2} \cdot \ln\left(\frac{1 + 0}{1 - 0}\right) = 0$$  

In order to derive the probability that an estimated Z-transformed correlation coefficient $z_{K=K_h}$ of the correct key guess is greater than a Z-transformed correlation coefficient $z_{K\neq K_h}$ of a wrong key guess, we need to evaluate their distance, i.e. $d = z_{K=K_h} - z_{K\neq K_h}$. As explained in [Ash93], the distance $d$ is also normally distributed with mean $\mu_d = \frac{1}{2} \cdot \ln\left(\frac{1 + \rho}{1 - \rho}\right)$ and variance $\sigma_d^2 = \frac{2}{N-3}$.

$$P(z_{K=K_h} > z_{K\neq K_h}) = P(d = z_{K=K_h} - z_{K\neq K_h} > 0)$$  
$$= 1 - P(d = z_{K=K_h} - z_{K\neq K_h} < 0)$$  
$$= \Phi \left( \frac{1}{2} \cdot \ln\left(\frac{1 + \rho}{1 - \rho}\right) \right)$$  

Solving this equation for the required number of measurements $N$ results in:

$$N = 3 + 8 \cdot \left( \frac{\Phi^{-1}(P(d > 0))}{\ln\left(\frac{1 + \rho}{1 - \rho}\right)} \right)^2$$  

As suggested in [Man04], a conservative confidence interval is $P(d > 0) = 0.9999$, which basically states that 99.99% of all correlation coefficients of wrong key guesses shall be less than the correlation coefficient of the correct key guess. Using the correlation coefficients given in Table 4.3 and formula 4.27, it is possible to assess the required number of measurements for HODPA attacks. These assessed measurement costs are given in Table 4.5. It is obvious that they are very similar to the experimentally derived numbers given in Table 4.4.

It must be pointed out that practical HODPA attacks will most certainly require more measurements. For example, the assumption that only 31 measurements are required to perform a first-order DPA is extremely optimistic and usually not achievable in a noisy measurement environment. In order to give a better estimation regarding the measurement costs we analyzed an 8051-based microcontroller whose power consumption behaviour matches surprisingly well the Hamming weight model. For this architecture the power leakage of some 8-bit variable $X$ at time $t_X$ can be modelled as

$$P(t_X) = offset + \epsilon \cdot W(X) + \sigma \cdot N.$$


<table>
<thead>
<tr>
<th>DPA Order $d$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>$3.75 \cdot 10^3$</td>
<td>$3.83 \cdot 10^5$</td>
<td>$3.49 \cdot 10^7$</td>
<td>$2.98 \cdot 10^9$</td>
<td>$2.44 \cdot 10^{11}$</td>
</tr>
</tbody>
</table>

Table 4.5.: Assessed number of measurements of HODPA attacks using the Fisher Z-transformation and the confidence interval $P(d > 0) = 0.9999$. The leakage is presumed to be equal to the Hamming weight of intermediate variables (no noise).

<table>
<thead>
<tr>
<th>DPA Order $d$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>$225.85$</td>
<td>$12539.1$</td>
<td>$3527564$</td>
</tr>
<tr>
<td></td>
<td>$\approx 2.26 \cdot 10^2$</td>
<td>$\approx 1.25 \cdot 10^4$</td>
<td>$3.52 \approx 10^6$</td>
</tr>
</tbody>
</table>

Table 4.6.: Number of measurements $N$ required to achieve an $|SNR|$ of $\geq 5$ in simulated HODPA based on the HW-model (parameters: $Offset = 10$ mA, $\epsilon = 3.72$ mA, $\sigma = 1.9636$ mA) attacks (averaged over 100 simulated DPA attacks for each order $d$).

In an experiment we analyzed $256 \cdot 1000 = 256000$ power traces and determined an average $offset = 10$ mA, current gain $\epsilon = 3.72$ mA, and Gaussian noise with a standard deviation $\sigma = 1.9636$ mA and $N \sim N(0, 1)$. Using these parameters we simulated 8 100 DPA attacks for each order $d = 1, \ldots, 4$ in order to determine the average measurement costs required to achieve an $|SNR| \geq 5$. The results are listed in Table 4.6.

As a result of the additive Gaussian noise $\sigma \cdot N$, the measurement costs roughly increase by a factor of $\approx 10$. Hence, the use of a noise generator as an add-on countermeasure is certainly reasonable to make DPA attacks more difficult [Man04]. In Figures 4.6 and 4.7 two correlation plots of a simulated second and third-order DPA are shown for the aforementioned parameters.

### 4.4.4. Single-Bit HODPA of the AES S-box Output

In the previous sections, we proposed theoretical results of HODPA attacks against hardware architectures with regard to the Hamming weight model. However, as discussed in [CJR+99a, ABDM00, LSP04], this model is of limited use in real-world attacks. In [Mes00b], a more general model was presented which focuses on a single bit and comprehends all remaining noise sources as a Gaussian distributed random variable. According to this model the two possible probability distributions of a power signal

---

8Real-world HODPA attacks against the 8051 microcontroller would have been possible, but with regard to the high measurement costs for orders $d > 2$ we decided to simulate these attacks.

9i.e. both arithmetic noise and measurement noise
4.4 Higher Order Masking of AES

Figure 4.6.: Correlation plot of a simulated second-order DPA against the AES S-box output according to the HW-model (parameters: $Offset = 10$ mA, $\epsilon = 3.72$ mA, $\sigma = 1.9636$ mA).

Figure 4.7.: Correlation plot of a simulated third-order DPA against the AES S-box output according to the HW-model (parameters: $Offset = 10$ mA, $\epsilon = 3.72$ mA, $\sigma = 1.9636$ mA).

$P(t_i)$ are defined as

$$f(P(t_i)|b = 0) \sim N(-\epsilon, \sigma^2) \quad \text{and} \quad f(P(t_i)|b = 1) \sim N(\epsilon, \sigma^2) \quad (4.28)$$

depending on the state of some bit $b$, e.g. an S-box output bit, which leaks at time $t_i$.

Let us assume the adversary measures $d$ power signals $P(t_i)$ which leak according to the general model

$$P(t_1) = \left(2\left(\begin{array}{c} \left\{S(X \oplus K) \oplus M\right\}[j] - 1 \end{array}\right) \right) + \sigma N_1$$

$$P(t_2) = \left(2\left(\begin{array}{c} \left\{M_1[j] - 1 \end{array}\right) \right) \quad \epsilon + \sigma N_2$$

$$\vdots$$

$$P(t_d) = \left(2\left(\begin{array}{c} \left\{M_{(d-1)}[j] - 1 \end{array}\right) \right) \epsilon + \sigma N_d$$

with $M = M_1 \oplus \ldots \oplus M_{(d-1)}$ and $N_1, \ldots, N_d \sim N(0, 1)$ and $0 \leq j \leq 7$. The correlation coefficient is defined as

$$\rho\left(\prod_{i=1}^{d} P(t_i), S(X \oplus K_h)[j] \right) = \frac{E[\prod_{i=1}^{d} P(t_i)S(X \oplus K_h)[j]] - E[\prod_{i=1}^{d} P(t_i)]E[S(X \oplus K_h)[j]]}{\sqrt{\prod_{i=1}^{d} P(t_i)|V[S(X \oplus K_h)[j]]}}$$
where \( S(X \oplus K_h)[j] \) denotes the state of bit \( j \) of a hypothesized S-box output \( S(X \oplus K_h) \). The expectation values in the numerator are

\[
E[\prod_{i=1}^{d} P(t_i) S(X \oplus K_h)[j]]
\]

\[
= E\left[ \sum_{S(X \oplus K) \oplus M = 0} P(t_1) \sum_{M_1 = 0}^{1} P(t_2) \ldots \sum_{M_{d-1} = 0}^{1} P(t_d) \sum_{S(X \oplus K_h)[j] = 0}^{1} S(X \oplus K_h)[j] \right]
\]

\[
= 2^{-(d+1)} \left( (-\epsilon) + \epsilon \right) 2^{(0+1)} = 0 \quad \text{if} \quad K_h \neq K
\]

\[
= 2^{-(d+1)} \left( \frac{2^d}{2} \right) (-1)^{d+1} \quad \text{if} \quad K_h = K
\]

The variances in the denominator are

\[
V[S(X \oplus K_h)[j]] = 0.25
\]

\[
V[\prod_{i=1}^{d} P(t_i)] = E[\prod_{i=1}^{d} P^2(t_i)] - E[\prod_{i=1}^{d} P(t_i)]^2
\]

\[
= \prod_{i=1}^{d} E[P^2(t_i)] = \prod_{i=1}^{d} E[(\epsilon + \sigma N)^2] = \prod_{i=1}^{d} \left[ E[\epsilon^2 + 2\epsilon \sigma N + \sigma^2 N^2] \right]
\]

\[
= (\epsilon^2 + \sigma^2)^d \quad \text{with} \quad N \sim N(0, 1) \quad \text{and} \quad \chi^2 = N^2 \sim \chi^2(1, 2)
\]

This results in the correlation coefficient

\[
\rho \left( \prod_{i=1}^{d} P(t_i), S(X \oplus K_h)[j] \right) = -\left( \frac{-\epsilon}{\sqrt{\epsilon^2 + \sigma^2}} \right)^d \quad \text{if} \quad K_h = K \quad (4.29)
\]

In order to estimate the correlation coefficient for various orders \( d \) we measured 1000 power traces from a test device\(^{10}\). Using this set of measurements we analyzed the power consumption caused by S-box 0 output bit 0 in round one and determined a mean \( \epsilon = 3.1838 \) mA and a standard deviation \( \sigma = 16.9143 \) mA. Using these parameters, we were able to estimate the correlation coefficients of DPA attacks for various orders \( d \). These numbers are listed in Table 4.7.

\(^{10}\)A smart card which is based on the AVR architecture and runs a software implementation of AES.

From our measurements we derived that this architecture does not agree very well with the Hamming weight model.
4.4 Higher Order Masking of AES

<table>
<thead>
<tr>
<th>DPA Order $d$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corr. Coeff. $\rho$</td>
<td>0.185</td>
<td>-0.0342</td>
<td>$6.30 \cdot 10^{-3}$</td>
<td>$-1.20 \cdot 10^{-3}$</td>
<td>$2.17 \cdot 10^{-4}$</td>
<td>$-4.01 \cdot 10^{-5}$</td>
</tr>
</tbody>
</table>

Table 4.7.: Correlation coefficients of a successful single-bit HODPA for various orders $d$ with parameters $\epsilon = 3.1838$ mA and $\sigma = 16.9143$ mA according to the general model.

<table>
<thead>
<tr>
<th>DPA Order $d$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>801.8</td>
<td>22614.37</td>
<td>1291118.02</td>
<td>17705001.01</td>
</tr>
<tr>
<td></td>
<td>$\approx 8.02 \cdot 10^2$</td>
<td>$\approx 2.26 \cdot 10^4$</td>
<td>$\approx 1.29 \cdot 10^6$</td>
<td>$\approx 1.77 \cdot 10^7$</td>
</tr>
</tbody>
</table>

Table 4.8.: Number of measurements $N$ required to achieve an $|SNR|$ of $\geq 5$ in simulated single-bit HODPA attacks with parameters $\epsilon = 3.1838$ mA and $\sigma = 16.9143$ mA (averaged over 100 simulated DPA attacks for each order $d$).

As in the previous section, we also performed simulated DPA attacks for various orders $d$ in order to determine the average number of measurements required to extract the correct key, i.e. to achieve an $|SNR| \geq 5$. These numbers are given in Table 4.8 and again show an exponential increase. Finally, in Figures 4.8 and 4.9 two correlation plots of a simulated second and third-order DPA are shown.

Using the correlation coefficients given in Table 4.7 and formula 4.27, it is again possible to assess the required number of measurements for HODPA attacks. These assessed measurement costs are given in Table 4.9. It is obvious that they are very similar to the experimentally numbers given in Table 4.8.

### 4.4.5. Secure HODPA AES Masking scheme

In this section, we propose an AES masking scheme which is secure against HODPA attacks. We assume that the adversary knows the exact points in time when any occurring intermediate variable leaks in the side channel trace and that she/he is able to measure

<table>
<thead>
<tr>
<th>DPA Order $d$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>793</td>
<td>$2.36 \cdot 10^4$</td>
<td>$6.97 \cdot 10^5$</td>
<td>$1.92 \cdot 10^7$</td>
<td>$5.87 \cdot 10^8$</td>
<td>$1.72 \cdot 10^{10}$</td>
</tr>
</tbody>
</table>

Table 4.9.: Assessed number of measurements of HODPA attacks using the Fisher Z-transformation and the confidence interval $P(d > 0) = 0.9999$. The leakage is presumed to depend on the state of a single bit, only, with additive Gaussian noise.
the corresponding power signal.

**Definition** A masking scheme which applies \((d-1)\) independent, random masks to blind a subkey-dependent intermediate variable is considered secure, if an adversary must perform an DPA attack of order \(d\), i.e. she/he must correlate at least \(d\) power signals with a selected function of the subkey-hypothesis, in order to successfully determine the secret subkey.

Let us consider a very simple and naive masking scheme based on a modified S-box \(S^*\) which is shown in Figure 4.10. We assume that the same set of \(d-1\) input and output masks \(M_1, ..., M_{d-1}\) are used to thwart DPA attacks up to order \(d - 1\). Unfortunately, this scheme has several vulnerabilities. First, note that the \(d-1\) masks at the S-box output can be regarded as a single x-or mask \(M = M_1 \oplus ... \oplus M_{d-1}\). While the x-or sum \(M\) may never leak by itself as an intermediate variable in the side channel trace, the variables \(K \oplus M\), \(X \oplus K \oplus M\) and \(S(X \oplus K) \oplus M\) do occur and thus cause a leakage. We observed that this gives rise to the following two counterintuitive second-order attacks
even if \( d - 1 > 1 \) masks \( M_i \) are used. The two correlation coefficients

\[
\rho \left( W(S(X \oplus K) \oplus M) \cdot W(K \oplus M), W(S(X \oplus K_h) \oplus K_h) \right)
\]

and

\[
\rho \left( W(S(X \oplus K) \oplus M) \cdot W(X \oplus K \oplus M), W(S(X \oplus K_h) \oplus X \oplus K_h) \right)
\]

will result in distinct peaks, if the correct key hypothesis is guessed. A simple way to
thwart both attacks is to use a different set of input and output masks for an S-box.
Furthermore, let us assume that different input and output masks are used for an S-box,

\[
\begin{align*}
\text{Figure 4.10: Insecure AES masking scheme using the same } d - 1 \text{ input and output masks to thwart DPA attacks of order } d.
\end{align*}
\]

however, the same two sets of \( d - 1 \) input masks \( M_i \) and output masks \( N_i \) are used for
all S-boxes. As suggested in [GT02], this leads to the following second-order attack with

\[
\rho \left( W(S(X \oplus K_N) \oplus N)W(S(Y \oplus K_N) \oplus N), W(S(X \oplus K_{HX})W(S(Y \oplus K_{HY})) \right)
\]

where \( N = N_1 \oplus ... \oplus N_{d-1} \) denotes the x-or sum of the output masks, \( X,Y \) denote two
arbitrary plaintext bytes, \( K_X,K_Y \) the two corresponding key bytes in the first round
and \( K_{HX},K_{HY} \) the two corresponding key hypotheses guessed by the adversary. Thus,
the hypothesis space is increased to 16 key bits which is still feasible. An insufficient
measure to counteract this second-order attack would be the random permutation\(^\text{11}\) of
the 16 S-boxes in each round, since this would merely increase the measurement costs
by a factor of \( 16 \cdot 15 \cdot \frac{1}{2} = 120. \) A better countermeasure is the usage of different input
and output masks for each S-box.

**Design Rule** Every AES S-box \( S_j^* \) with \( 1 \leq j \leq 16 \) should use a different set of
\( d - 1 \) input masks \( M_{(j,1)}, \ldots, M_{(j,d-1)} \) and output masks \( N_{(j,1)}, \ldots, N_{(j,d-1)} \) for each round
to thwart DPA attacks of orders \( < d. \)

\(^{11}\) i.e. a temporal desynchronisation of the power traces
Figure 4.11.: Secure AES masking scheme which uses $d - 1$ different input and output masks for each S-box to thwart DPA attacks of order $d$.

### 4.4.6. S-box Recomputation Algorithm

In the case of AES, 8-bit x-or masks are used to blind elements in $GF(2^8)$. As stated in [Mes00b, AG01, GT02], the only transformation in AES which requires special attention with regard to masking is the non-linear S-box, which performs an inversion in $GF(2^8)$ followed by an affine bitwise transformation. For this reason, an x-or mask $M$ will not propagate unchanged through the S-box.

$$s((X \oplus K) \oplus M) = s(X \oplus K) \oplus R \neq s(X \oplus K) \oplus M \quad \text{for any } X \oplus K, M \neq \{0\}$$

The S-box must be modified in such a way that $s((X \oplus K) \oplus M) = s(X \oplus K) \oplus M$ for $\forall X \oplus K$. This can be achieved twofold: either by simple recomputation of the S-box [Mes00a] or by algebraic methods [BGK04, OS05, CG05, RS05]. The disadvantage of algebraic methods is that they are usually not very efficient when implemented in software and generally do not address higher order masking. In [Mes00a], a very simple recomputation algorithm was proposed which blinds the index of a table $S$ with mask $M$, the output with mask $N$ and stores it as a new table $S'$. This algorithm was already discussed in Section 4.1. It requires 256 read and write instructions and 512 bytes RAM\(^{12}\) for tables $S$ and $S'$. In [TSG02], Trichina et al. suggested the "split and swap" algorithm which is denoted as algorithm 2. The initial step applies the output mask $N$ and requires 256 read and write instructions. If a bit $M[j]$ is set, $2^{s-(j+1)}2^{j+1} = 256$ read and write instructions are required for each split-and-swap operation. This results in an average total of $256 + 4 \cdot 256 = 1280$ read and write instructions. As an advantage, only 256 bytes of RAM are required to recompute the S-box. As an alternative we propose the S-box recomputation algorithm 3 which only requires 256 read and write instructions and only 256 bytes of RAM. In Table 4.10, the performance of the three S-box masking algorithms is compared.

\(^{12}\)We will see later that in the case of higher order masking it is inefficient to store table $S$ in ROM and only $S'$ in RAM even though it makes sense to do so in the case of first-order masking.
Algorithm 2 Computation of the Masked AES S-box as proposed by Trichina et al. in [TSG02]

Require: $M, N$
Ensure: $S^*(X \oplus M) = S(X) \oplus N$,
1: for $i = 0$ to $255$ do
2: $S^*(i) = S(i) \oplus N$
3: end for
4: for $j = 0$ to $7$ do
5: if $M[j] = 1$ then
6: (1) Split $S^*$ into succeeding blocks of $2^i$ elements
7: (2) Swap pairwise the $(2n)$th and $(2n+1)$th,... block
8: end for
9: Return($S^*$)

Algorithm 3 Our proposed S-box masking algorithm

Require: $M, N, j$ (=index of the most significant bit set in $M$)
Ensure: $S^*(X \oplus M) = S(X) \oplus N$,
1: for $i = 0$ to $255$ with $i = i + 2^{j+1}$ do
2: for $l = 0$ to $2^j - 1$ do
3: $A = S(i \oplus l)$
4: $B = S(i \oplus l \oplus M)$
5: $S^*(i \oplus l) = B \oplus N$
6: $S^*(i \oplus l \oplus M) = A \oplus N$
7: end for
8: end for
9: Return($S^*$)

<table>
<thead>
<tr>
<th>S-box masking algorithm</th>
<th>#read/write cycles</th>
<th>#RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Messerges et al. [Mes00a]</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>Trichina et al. [TSG02]</td>
<td>1280</td>
<td>256</td>
</tr>
<tr>
<td>Our proposed algorithm</td>
<td>256</td>
<td>256</td>
</tr>
</tbody>
</table>

Table 4.10.: Overview of different S-box masking algorithms.
Let us assume we would want to apply $d - 1$ input masks $M_1, ..., M_{d-1}$ and $d - 1$ output masks $N_1, ..., N_{d-1}$ to the original S-box $S$ which has been copied from ROM into RAM. Since the x-or sums $M = M_1 \oplus ... \oplus M_{d-1}$ and $N = N_1 \oplus ... \oplus N_{d-1}$ shall never leak during the execution of the cipher, one possibility is to recompute the S-box $d - 1$ times:

$$
\text{recompute}(S, M_1, N_1) \rightarrow ... \rightarrow \text{recompute}(S, M_{d-1}, N_{d-1})
$$

The order of the recombination steps is arbitrary. Fortunately, it is only necessary to perform these $d - 1$ recomputations for the very first S-box in round one. Once the first S-box is masked with $M$ and $N$, it is easy to derive a new S-box with input masks $U_1, ..., U_{d-1}$ and output masks $V_1, ..., V_{d-1}$ by using the chain of masks $U'$ and $V'$.

$$
U' = U_1 \oplus M_1 \oplus ... \oplus U_{d-1} \oplus M_{d-1}
$$

$$
V' = V_1 \oplus N_1 \oplus ... \oplus V_{d-1} \oplus N_{d-1}
$$

Thus, the x-or sum $U'$ removes the previous input masks $M_i$ and adds the new input masks $U_i$, while the x-or sum $V'$ removes the previous output masks $N_i$ and adds the new output masks $V_i$ in one step. It is important that the previous and new masks are stacked up in the alternating order given above to avoid any possible side channel vulnerabilities. As a result, only a single recombination step $\text{recompute}(S, U', V')$ is required to derive the new S-box independent of the number of masks $d - 1$.

### 4.4.7. Mask Propagation and the MixColumn Transformation

For AES implementations which must be secure against first-order DPA, only, it is sufficient to use a single 8-bit mask $M$ for the entire algorithm. As a matter of fact, the mask $M$ will simply propagate through the MixColumn transformation and no attention must be paid to correct the mask after it has propagated through the MixColumn transformation\(^{13}\).

$$
\text{MixCol} \left( \begin{array}{c} S(X_1 \oplus K_1) \oplus M \\ S(X_2 \oplus K_2) \oplus M \\ S(X_3 \oplus K_3) \oplus M \\ S(X_4 \oplus K_4) \oplus M \end{array} \right) = \text{MixCol} \left( \begin{array}{c} S(X_1 \oplus K_1) \\ S(X_2 \oplus K_2) \\ S(X_3 \oplus K_3) \\ S(X_4 \oplus K_4) \end{array} \right) \oplus \left( \begin{array}{c} M \\ M \\ M \\ M \end{array} \right)
$$

With regard to an AES implementation resistant against a DPA attack of order $d$ let us assume that $d - 1$ different input masks $M(j,1), ..., M(j,d-1)$ and $d - 1$ different output masks $N(j,1), ..., N(j,d-1)$ are used for each S-box $j$ in the first round with $M_j = M(j,1) \oplus$

\(^{13}\)Please note that special care has to be taken when a single mask $M$ is used in connection with the MixColumn transformation. The computation of the MixColumn transformation must be performed in a carefully chosen order so that the mask $M$ is never cancelled out at any time.
\[ ... \oplus M_{(j, d-1)} \text{ and } N_j = N_{(j,1)} \oplus ... \oplus N_{(j, d-1)} \text{ and } 1 \leq j \leq 16. \text{ In this case, the masks do change after they have propagated through the MixColumn transformation.} \]

\[
\begin{pmatrix}
S(X_1 \oplus K_1) \oplus N_1 \\
S(X_2 \oplus K_2) \oplus N_2 \\
S(X_3 \oplus K_3) \oplus N_3 \\
S(X_4 \oplus K_4) \oplus N_4
\end{pmatrix} = \begin{pmatrix}
S(X_1 \oplus K_1) \\
S(X_2 \oplus K_2) \\
S(X_3 \oplus K_3) \\
S(X_4 \oplus K_4)
\end{pmatrix} \oplus \begin{pmatrix}
N'_1 \\
N'_2 \\
N'_3 \\
N'_4
\end{pmatrix}
\]

In order to follow the propagation of the output masks \(N_{(j,1)}, \ldots, N_{(j,d-1)}\), the MixColumn transformation must be executed an additional \(d - 1\) times for each column.

\[
\begin{pmatrix}
N'_{(1,1)} \\
N'_{(2,1)} \\
N'_{(3,1)} \\
N'_{(4,1)}
\end{pmatrix} = \begin{pmatrix}
N_{(1,1)} \\
N_{(2,1)} \\
N_{(3,1)} \\
N_{(4,1)}
\end{pmatrix} \oplus \begin{pmatrix}
N'_{(13,d-1)} \\
N'_{(14,d-1)} \\
N'_{(15,d-1)} \\
N'_{(16,d-1)}
\end{pmatrix}.
\]

For example, in an implementation secure against second-order DPA attacks, the MixColumn transformation must be executed an additional \(4 \cdot 2 = 8\) times.

### 4.4.8. HODPA-Resistant AES Implementations

We implemented the following AES implementations on an AVR-based smart card in assembly: an unmasked AES implementation, a first-order DPA-resistant AES implementation using a single mask for the entire AES and, finally, implementations resistant to second, third and fourth-order DPA using different input and output masks for all S-boxes. The details such as code sizes and data sizes of these implementations are given in Table 4.11. Moreover, the number of cycles required for an encryption and the corresponding execution times\(^{14}\) are given, as well.

Due to the diffusion characteristics of the MixColumn transformation [DR02], an S-box output in round two depends on 32 key bits and in round three already on 128 key bits. Because of performance issues, we only masked the first three and the last three rounds in our AES implementations. Furthermore, we developed two sets of AES implementations resistant to HODPA. The first set listed in Table 4.11 uses the simple S-box recomputation algorithm suggested in [Mes00a] which requires 512 bytes of RAM but is quick. The second set listed in Table 4.11 uses our proposed S-box recomputation algorithm which requires only 256 bytes of RAM but could not be implemented.

\(^{14}\) under the assumption that the device is clocked at 5 MHz
<table>
<thead>
<tr>
<th>DPA resistance</th>
<th>S-box algo.</th>
<th>code size [bytes/ROM]</th>
<th>data size [bytes/RAM]</th>
<th>cycles</th>
<th>time [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>unprotected 1st order resistant</td>
<td>- [Mes00a]</td>
<td>1078</td>
<td>16</td>
<td>4625</td>
<td>0.925</td>
</tr>
<tr>
<td>2nd order resistant</td>
<td>[Mes00a]</td>
<td>2798</td>
<td>592</td>
<td>193199</td>
<td>38.6</td>
</tr>
<tr>
<td>3rd order resistant</td>
<td>[Mes00a]</td>
<td>3350</td>
<td>624</td>
<td>197263</td>
<td>39.5</td>
</tr>
<tr>
<td>4th order resistant</td>
<td>[Mes00a]</td>
<td>3962</td>
<td>656</td>
<td>201255</td>
<td>40.2</td>
</tr>
<tr>
<td>2nd order resistant</td>
<td>see 4.4.6</td>
<td>2614</td>
<td>336</td>
<td>243581</td>
<td>48.7</td>
</tr>
<tr>
<td>3rd order resistant</td>
<td>see 4.4.6</td>
<td>3164</td>
<td>368</td>
<td>247573</td>
<td>49.5</td>
</tr>
<tr>
<td>4th order resistant</td>
<td>see 4.4.6</td>
<td>4174</td>
<td>400</td>
<td>260229</td>
<td>52.0</td>
</tr>
</tbody>
</table>

Table 4.11.: Details of various HODPA resistant AES AVR implementations.

as efficiently in assembly as the simple S-box recomputation algorithm due to pointer arithmetic issues.
5. Template-Enhanced DPA

In this chapter, we will discuss two new applications of template attacks: single-bit template classification and a method to break the masking countermeasure based on single-bit template classification. We begin with a brief overview of previous works on template attacks in Section 5.1. The novel contributions of this chapter are presented in Sections 5.2 and 5.3. In Section 5.2 we show that a single side channel trace does not only contain enough information to reveal the state of a single byte, but even the state of a single bit can be classified with low error probability, if multivariate signal analysis is applied. Based on this single-bit classification, a new attack is presented in Section 5.3, which can defeat the masking countermeasure [GP99, AG01] under certain conditions. This presumes that the adversary has access to a smart card or cryptographic token which is protected by the masking countermeasure, but whose random number generator is bust or has some imperfect bias. The adversary can then build templates which allow to break an identical smart card even if its RNG is perfectly functional and has no bias. Thus, this attack opens up a potential back door by giving smart card manufacturers, vendors and developers the possibility to break their own implementations, if they have access to a card, whose RNG is biased, bust or has been intentionally destroyed. As a proof of concept, we show in Section 5.3.3 the results of an attack against a masked implementation of the DES running on a 6805-based smart card a masked implementation of the AES running on an AVR-based smart card. Parts of this work were published in 2005 at the Cryptographic Hardware and Embedded Systems (CHES) conference in [ARRS05].

5.1. Previous Work

Template attacks are based on multivariate signal classification which analyzes the pairwise covariances of a chosen subset of significant points in a side channel trace. Ideally, this subset embraces all those points in a trace, which are strongly related with the state of a chosen key-dependent intermediate variable, e.g. an S-box output byte in an AES implementation. The usage of multivariate statistics in side channel cryptanalysis was introduced by Chari et al. as template attacks in [CRR02]. Template attacks generally consist of two steps: first, an adversary derives covariance models, i.e. templates, from
a fully accessible test device\(^1\), then, he/she uses these templates with an identical, but not fully accessible target device\(^2\) to classify the state of a chosen key-dependent intermediate variable. Depending on the quality of the templates, only a single side channel trace of the target device must be analyzed, in order to extract parts of the secret key with reasonable high probability [CRR02, RO04]. As already discussed in Section 2.7, each template \(i\) consists of a mean vector of significant points \(\overline{T}_i\) and a covariance matrix \(C_i\) of these points. The vector of means can be understood as the part of the template which describes the signal whereas the covariance matrix characterizes the noise. Template attacks evaluate a multivariate probability density function which gives the likelihood that a side channel trace \(I'\) observed from a target device matches a template \((\overline{T}_i, C_i)\) acquired from an identical test device.

\[
p(N'_i) = \frac{1}{\sqrt{(2\pi)^L \cdot |C_i|}} \cdot \exp \left( -\frac{1}{2} \cdot N'_i^T \cdot C_i^{-1} \cdot N'_i \right) \tag{5.1}\]

where \(N'_i = I' - \overline{T}_i\) denotes the noise vector of the observed trace wrt. template \(i\). If, for instance, an adversary wanted to classify the state of the output bit 0 of S-box 1 of a DES implementation in round one, he/she would have to examine how well the observed trace \(I'\) matches the two templates \((\overline{T}_0, C_0)\) and \((\overline{T}_1, C_1)\) corresponding to the state of the bit. As shown in [ARR03], this decision problem can be reduced to the following inequality.

\[
(I' - \overline{T}_0)^T C_0^{-1} (I' - \overline{T}_0) - (I' - \overline{T}_1)^T C_1^{-1} (I' - \overline{T}_1) \geq \ln (|C_1|) - \ln (|C_0|) \tag{5.2}\]

where a decision is made in favor of template \((\overline{T}_i, C_i)\), if the inequality holds true. If the classified variable has more than two outcomes, the decision problem is solved in a maximum likelihood approach. An interesting case occurs, if \(C = C_0 = C_1\), i.e. the noise is data-independent. Then, Charli et al. state in [CRR02] that the probability of error in the maximum likelihood test is

\[
P_e = \frac{1}{2} \cdot \text{erfc} \left( \frac{1}{2} \cdot \sqrt{\frac{(\overline{T}_1 - \overline{T}_0)^T C^{-1} (\overline{T}_1 - \overline{T}_0)}{2}} \right) \tag{5.3}\]

where \(\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt\) (see [Funal]).

In [CRR02], Charhi et al. pointed out that compared to classical univariate methods (e.g. SPA, DPA) multivariate signal analysis is a very efficient classification method to extract secret key data from ciphers, where only a limited number of side channel traces

\(^1\)It is presumed that the adversary is not only able to collect side channel traces from the test device, but also knows precise details such as the timing behavior of the device under test.

\(^2\)It is only presumed that the adversary is able to collect side channel traces from the target device, whose key is the secret.
with low signal-to-noise ratio is available. As a matter of fact, they were able to extract secret key bytes from an RC4 software implementation using a single power trace.

In [ARR03], Agrawal et al. showed that template attacks can be further improved by concatenating traces of multiple channels, e.g., EM emissions at different carrier frequencies, to a single trace. However, Agrawal et al. also warned that the selection of source channels is often very tricky and counter-intuitive.

In [RO04], Rechberger and Oswald pointed out that the proper selection of significant points is a crucial factor in template attacks. Since templates classify a particular state of a processor, such as the value of a key-dependent run-time variable, it is important to choose significant points of high variance with regard to the possible states of the variable of interest. As pointed out in [BNSQ03], one approach to find these points is to apply a Principal Component Analysis (PCA) of the set of measured side channel traces. PCA reduces the dimension of the data set and preserves points of maximum variance. However, one major disadvantage of PCA is its high computational cost. In the context of template attacks, in [RO04] Rechberger and Oswald suggest a simpler and computational less expensive approach to find significant points of high variance, which resembles classical DPA and examines the differences of means of the sets of measurements. Moreover, they examined the classification results for various numbers of significant points and various minimum temporal distances between these points. They found out that template classification performance decreases as the number of significant points exceeds an implementation-specific threshold. Moreover, they demonstrated that preprocessing of side channel traces during the training and classification phase, e.g., the application of a Fast Fourier Transformation (FFT), can further increase the classification results of template attacks.

5.2. Single-Bit Template Classification

All previously published works [CRR02, ARR03, RO04] used template attacks to classify the state of a byte, e.g., a byte of the 256-byte initial state table in the stream cipher RC4. However, in this section we will show that template attacks can also be used to reveal the state of a single bit with only a single side channel trace measured from a target device. This is not as straightforward as it seems, since simultaneous bit switching activities further increase the noise floor and, thus, make a classification more difficult. For example, let us consider the S-box output bits of DES in round one. If we build a pair of templates for each S-box output bit corresponding to the zero- and one-state and classify each bit with an average success rate $\eta_{S_{bj}}$ with $1 \leq i \leq 8$ and $0 \leq j \leq 3$, all 32 bits of the first round key can be extracted with certain error. The remaining 24 key bits can then be easily found with an exhaustive search or by template classification of the S-box outputs in the last round when ciphertexts are decrypted. In order to verify, whether single-bit template classifications are practical, we implemented DES in assembly on a
smart card $A^3$, and collected 1400 power traces from the device during encryptions of random plaintext data. Then, we performed a void hypothesis DPA (see Section 2.5) for each S-box output bit in order to identify significant points in the power traces for the subsequent construction of templates. As argued in [ARR03], the void hypothesis DPA is better suited than standard DPA, because it also takes signal variances into account. In Figure 5.1, the improved metric of S-box 1, bit 0 is shown. Metric 5.1 reveals that several points in a side channel trace are related to the state of output bit 0 of S-box 1. In case of the observed DES software implementation, the S-box output bit leaks at multiple points in time, because the run-time variable, which stores the S-box output bit, is accessed several times during the 32-bit permutation $P$ following the DES S-boxes. In general, cryptographic algorithms with low diffusion properties are ideal candidates for multivariate single-bit classification, because an observed bit is likely to leak at several times.

We selected 50 significant points from the 32 DPA metrics and built a pair of templates for each S-box output bit. Then, we collected an additional set of 100 power traces from smart card $A$ and classified the states of the 32 S-box output bits with the $32 \cdot 2 = 64$ templates, collected from the same device. The classification success rates $\eta_{S,b}$, which state how often an S-box output bit was classified correctly are listed in Table 5.1. Note that the classification rates range from 0.72 to 1.00 which means that in the worst case\footnote{\textsuperscript{3}Smart card $A$ is based on a Motorola 6805 architecture.} 72\% of the states of an S-box output bit were classified correctly. Hence, the estimated probability that the entire 32-bit output of all S-boxes is classified correctly from a single

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure5.1}
\caption{Improved DPA metric of S-box 1, bit 0 of a test device (smart card $A$) running DES.}
\end{figure}

\textsuperscript{4}In our experiments S-box 3/bit 3 and S-box 6/bit 0
Table 5.1.: S-box output bit classification success rates $\eta_{S,b_j}$ using templates built with 1400 samples and 50 significant points.

\[
P = \prod_{i=1}^{8} \prod_{j=0}^{3} \eta_{S,b_j} << \min(\eta_{S,b_j})
\]  

Even though it is not possible to exactly classify the 32-bit S-box output and correctly determine the entire round key, the costs of a brute force attack are decreased significantly by assigning each 6-bit subkey with a weightening factor depending on its classification rate\(^5\). It is also possible to further optimize the attack by directly classifying key bits instead of S-box output bits, because key bits will leak at several times, e.g. during the key schedule algorithm. Hence, templates are capable to classify the state of a single bit in a side channel trace with reasonable probability, despite the fact that the leakage of this bit is superimposed by several sources of noise and remaining bits, which leak concurrently.

5.3. Breaking the Masking Countermeasure: Template-Enhanced DPA

5.3.1. Overview

The proposed attack consists of two steps: a profiling phase and a hypothesis testing phase. During the profiling phase an adversary performs a DPA attack against a cipher running on a test device which is protected by the masking countermeasure (see Chapter 4). In our adversary model we presume that the RNG which generates the random

\(^{5}\) i.e. if the four output bits of an S-box are classified with low error probability, the corresponding 6-bit subkey is less likely to change during a brute force attack than a 6-bit subkey which corresponds to an S-box output that was classified with high error probability.
masks is not perfectly functional and has some unknown bias. Under this condition a DPA of the test device, e.g. focussing on an S-box output bit, will be successful with a certain number of measurements and will thus result in a differential trace which indicates significant points, when the masked S-box output bit leaks\(^6\). As discussed in Section 5.2, once significant points have been identified it is possible to build templates. However, please note that the templates will contain an RNG bias-dependent error and, thus, will not be perfect: the greater the off-bias of the RNG, the less the error in the templates.

During the hypothesis testing phase the adversary focuses on an identical target device and uses the previously built templates to classify the state of the masked S-box output bit. Please note that we do not presume that the RNG of the target device is faulty or has some kind of bias. However, during the profiling phase an error is induced into the templates depending on the bias of the test device which will result in a classification error during the hypothesis testing phase. In addition to template classification the adversary can also predict the state of the unmasked S-box output bit using a key hypothesis. As a result, the x-or sum of the masked S-box output bit classified with the pair of templates and the predicted unmasked S-box output bit reveals the mask bit itself which usually also leaks during the execution of the cipher. If the adversary made a correct key hypothesis when predicting the unmasked S-box output bit, then a DPA with the mask bit as a selection function will result in significant peaks. In a nutshell, our proposed template-enhanced DPA attack consists of the following steps:

1. Perform a single-bit DPA of a test device which is protected by the random masking countermeasure, but whose RNG has some bias. This will reveal points in time when the masked bit, e.g. a masked S-box output bit, leaks.

2. Assume that the RNG is bust\(^7\) and use the measured side channel traces to build templates which classify the masked S-box output bit with some RNG bias-dependent error.

3. Encrypt (decrypt) random plaintexts (ciphertexts) with an identical target smart card, whose RNG can be perfectly biased, measure the corresponding power traces and classify the masked S-box output bit for each trace.

4. Predict the unmasked S-box output bit for each trace using a key hypothesis and compute the mask for each trace.

5. Perform a DPA of the target device with the mask bit as a selection function. If the key hypothesis used in the previous step (4.) was correct, significant peaks will show up in the differential trace, when the mask bit leaks.

\(^6\)The greater the off-bias of the RNG from 0.5 the less measurements are required to successfully perform a DPA and the greater the (absolute) correlation of the occurring masked S-box output bit and the predicted unmasked S-box output bit.

\(^7\)Simply assume that additive x-or masks are always zero.
We will review these steps in more detail in the next two sections.

5.3.2. Profiling Phase

We assume the adversary or some entity has access to a protected smart card, whose RNG is biased, i.e., the random generation of zero and one bits is not uniformly distributed. Furthermore, we assume that the only protection an adversary has to face is the masking countermeasure, e.g., the duplication method published in [GP99], and that other countermeasures, such as desynchronization issues, have been removed and are not present. The masking countermeasures generally blinds all intermediate, key-dependent variables with randomly generated masks. Hence, the encryption of fixed plaintexts with a fixed key will result in different side channel traces. Depending on the algorithm intermediate variables are blinded with different masks, e.g. in a part of an algorithm which uses bitwise operations intermediate variables will be masked with an \( x \)-or mask, in a part of an algorithm which uses arithmetic additions intermediate variables will be masked with additive masks and so on [CC00]. The original value can be recovered from the blinded value by applying the inverse mask. However, non-linear functions, e.g., the S-boxes in DES and AES are an exception, because an inverse mask, which recovers the unmasked S-box output, cannot be computed from the input mask. As thoroughly discussed in Chapter 4, S-boxes, whose inputs and outputs are masked, are usually computed prior to each invocation of an algorithm and stored in RAM. As an example, a masked S-box is shown in Figure 5.2. Even though the term \( s(x \oplus k) \) usually denotes a multi-bit S-box output value, we will use it in the subsequent text to denote a single S-box output bit, e.g. bit 0 of S-box 1 in DES. Note that the unmasked S-box output \( s(x \oplus k) \) never occurs as a run-time variable during the execution of the algorithm, however, we presume that both the masked output \( s(x \oplus k) \oplus m \) and the mask \( m \) do occur and thus leak in the side channel trace.

\[
(x \oplus k) \oplus n \\
\text{masked S-box 1, bit 0} \\
\downarrow \\
s(x \oplus k) \oplus m \\
m
\]

Figure 5.2.: Blinded S-box input and output bit with a random input mask bit \( n \) and a random output mask bit \( m \).

Let the RNG of the device under test have some error \( \nu \), with \( 0 \leq \nu \leq 1 \). We define an
error of $\nu = 1$ as a bias of 100/0, i.e. the RNG is bust and only generates zero mask bits, an error of $\nu = 0.5$ corresponds to a perfectly functional bias, i.e. the generation of zero and one mask bits is uniformly distributed. An error of $\nu = 0$ corresponds to a bias of 0/100, i.e. the RNG is bust and generates only mask bits which are set.

If an adversary makes a correct key hypothesis $k_h = k$ and performs a DPA using bit $s(x \oplus k)$ as a selection function, he/she will assign the side channel traces $I_j(t)$ to the zero-partition and one-partition with error probability $P = 1 - \nu$. Hence, the incorrect assignment of side channel traces can be understood as an additional source of noise in the differential trace. Let the masked S-box output bit $s(x \oplus k) \oplus m$ leak at time $t_0$ and let $\epsilon$ denote its contribution to the power consumption, i.e. $I(t_0) = \epsilon \cdot (s(x \oplus k) \oplus m) + N$ where $N$ denotes additive Gaussian noise with mean $\mu_N$ and variance $\sigma_N^2$. As discussed in Section 2.2, the expected values of the zero-partition and one-partition at time $t_0$ are:

$$E[I_j(t_0)|s(x \oplus k) = 0] = \mu_n + (1 - \nu) \cdot \epsilon$$

$$E[I_j(t_0)|s(x \oplus k) = 1] = \mu_n + \nu \cdot \epsilon$$

The differential trace $\Delta(t_0)$ turns out to be

$$\Delta(t_0) = E[I_j(t_0)|s(x \oplus k) = 1] - E[I_j(t_0)|s(x \oplus k) = 0] = (2 \cdot \nu - 1) \cdot \epsilon$$

whereas $\Delta(t \neq t_0)$ approximates zero. A possible expression of the SNR of a differential trace in DPA attacks was given by Messerges et al. in [MDS99]. We enhance their SNR description with the additional noise factor $(2 \cdot \nu - 1)$ caused by the biased RNG, which yields

$$SNR = (2 \cdot \nu - 1) \cdot \epsilon \cdot \frac{\sqrt{M}}{\sigma_N}$$

where $M$ denotes the number of side channel traces. Let us assume that in case of an RNG bias at $\nu = 1$, an adversary would have to measure $N = 100$ traces to obtain a differential trace with a certain acceptable SNR. Table 5.2 lists the number traces, for various RNG biases $\nu$, in order to achieve an equal SNR. A successful DPA of the test device reveals points in time, when the masked S-box output bit $s(x \oplus k) \oplus m$ leaks. In the second step of the profiling phase these points are used to build a pair of templates in order to classify the state of the masked S-box output bit. When building the two templates the adversary blindly assumes that the RNG of the test device is biased at 100/0 ($\nu = 1$), i.e., the S-box output $s(x \oplus k)$ and the masked output $s(x \oplus k) \oplus m$ are fully correlated. However, this assumption is only true with probability $P = \nu$. Thus, $(1 - \nu) \cdot M$ traces are incorrectly used to build the pair of templates and a bias-dependent error is introduced into the templates.

Clearly, if the RNG is not fixed at 0, but has a probability $\nu$ of outputting a 0 bit, the templates built by an adversary have significant errors. For example when $\nu > 0.5$,
then the 0-bit template will be built using roughly $\nu \times N/2$ samples that are actually 0 samples and roughly $(1 - \nu) \times N/2$ samples that are actually 1's. When $\nu < 0.5$, then the templates are inverted; the 0 template is built using more 1 samples than 0 samples. Such templates are equally useful since they will consistently predict the bit incorrectly with high probability. When $\nu = 0.5$, DPA will not work and the templates as described here cannot be built.

We will show later in that even though significant errors are introduced in the templates when the RNG is very slightly biased, i.e., when $\nu$ is close to 0.5, if enough signals are used to build these templates, then the performance of the template-enhanced DPA attack is not significantly impacted—the attack works almost as well as an attack using perfect templates ($\nu = 1$).

### 5.3.3. Hypothesis Testing Phase

Let us assume the adversary has built approximated templates to classify a chosen masked S-box output bit using a test device, which has some unknown RNG bias $\nu$. If the adversary has access to an identical target device, he/she can use these templates to classify the state of the masked S-box output bit $s(x \oplus k) \oplus m$ with some classification error $\epsilon$. Furthermore, if the adversary makes a correct key hypothesis regarding the secret key $k$ used in the target device, he/she can predict the unmasked output bit

<table>
<thead>
<tr>
<th>RNG bias</th>
<th>$\nu$</th>
<th>$M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/0</td>
<td>1.00</td>
<td>100</td>
</tr>
<tr>
<td>95/05</td>
<td>0.95</td>
<td>123</td>
</tr>
<tr>
<td>90/10</td>
<td>0.90</td>
<td>156</td>
</tr>
<tr>
<td>85/15</td>
<td>0.85</td>
<td>204</td>
</tr>
<tr>
<td>80/20</td>
<td>0.80</td>
<td>278</td>
</tr>
<tr>
<td>75/25</td>
<td>0.75</td>
<td>400</td>
</tr>
<tr>
<td>70/30</td>
<td>0.70</td>
<td>625</td>
</tr>
<tr>
<td>65/35</td>
<td>0.65</td>
<td>1111</td>
</tr>
<tr>
<td>60/40</td>
<td>0.60</td>
<td>2500</td>
</tr>
<tr>
<td>55/45</td>
<td>0.55</td>
<td>10000</td>
</tr>
<tr>
<td>52/48</td>
<td>0.52</td>
<td>62500</td>
</tr>
<tr>
<td>51/49</td>
<td>0.51</td>
<td>250000</td>
</tr>
</tbody>
</table>
\[ s(x \oplus k) \text{ and, thus, the mask bit } m \text{ itself}^{8} \]

\[ m = \left[ s(x \oplus k) \right] \oplus \left[ s(x \oplus k) \oplus m \right] \tag{5.9} \]

Since the mask bit \( m \) is an intermediate variable in the algorithm, it will leak at some instance of time in the side channel trace. To summarize, the idea is to predict \( s(x \oplus k) \) and to classify \( s(x \oplus k) \oplus m \) for each trace, compute the mask bit \( m \) and use it as a selection function in a final DPA attack. If the attacker hypothesizes the correct subkey \( k \) and classifies the masked S-box output bit \( s(x \oplus k) \oplus m \) with success probability \( \eta \neq 0.5 \), peaks will show up in the corresponding differential trace at points in time, when the mask bit \( m \) leaks. The greater the template classification probability \( \eta \), the greater the SNR of the differential trace and thus, as shown in Table 5.2, the less measurements are required to perform the final DPA attack (please note that a little classification probability \( \eta < 0.5 \) will also result in a successful final DPA attack, since only the deviation \( |\eta - 0.5| \) is relevant).

### 5.3.4. Experimental Results

We performed the proposed template-extended DPA attack on two smartcards: a protected DES implementation on the smartcard A and a protected AES implementation on the smartcard B. For each smartcard, in the profiling phase, the templates were built with the RNG turned off (\( \nu = 1 \)). In the hypothesis testing phase, traces were obtained with the RNG on and working perfectly (\( \nu = 0.5 \)). For the smartcard A, the lower plot in Figure 5.3 shows the differential trace of the template-enhanced DPA attack on the hypothesized mask bit \( m \). A similar differential trace for the smartcard B is shown in the lower plot of Figure 5.4. Both plots contain distinct peaks even though the masking protection was fully functional. For completeness, Figure 5.5 shows a template-enhanced DPA trace for a false key hypothesis for smartcard B, which shows no peaks.

If the RNG of the test card during the profiling phase is just slightly biased instead of being broken, then the templates obtained from the test card would have significant cross-contamination. One may conjecture that, as a result, the classification rate \( \eta \) would be lower as the RNG bias \( \nu \) becomes smaller. However, this is not the case as we will show in Section 5.3.5.

**Theorem 5.3.1** If the noise covariance matrix of side channel traces is the same for two values of a mask bit and enough traces are available from a test card with a biased RNG \((0.5 < \nu < 1.0)\), then the templates prepared from such traces give the same probability of error as the templates obtained from a test-device with broken RNG \((\nu = 1)\).

---

8We assume that boolean masking is used.
In our experiments, we found that the noise covariance matrices of side channel traces for different values of mask bit are nearly the same. For the actual covariance matrices obtained in one of our experiments, we performed simulations of how well the signal classification works when templates are built using different numbers of samples from the test card with different RNG biases. In this simulations, the samples were generated by sampling from the noise probability distributions and the RNG bias was simulated by randomly misclassifying samples into the 0-partition and 1-partition used to build templates. We also performed an actual experiment where 1000 samples were obtained from the test card and templates were build for different RNG biases (again simulated by putting samples randomly in incorrect partitions). The results of these experiments are shown in Figure 5.6. Three plots are derived from the simulations involving 1000, 10,000, and 100,000 traces from a simulated test card with biased RNG to build templates. These three plots show that as the number of traces from the test card increases, the
probability of classification error becomes insensitive to the RNG bias. The fourth plot is the experimental using 1000 samples from a test card to build templates. The experimental curve is in excellent agreement with our analytical results.

In summary, even with a test card with very small RNG bias, it is possible to mount template-enhanced DPA attacks. The only effect of a small bias is that many more samples are needed to build templates that are as good as template built from a card with completely broken RNG.

### 5.3.5. Sensitivity of classification rate on RNG bias

In this section we will show that the template classification error is indeed independent of the RNG bias $\nu$ of the test device, and thus independent of the distortion in the templates,
5.3 Breaking the Masking Countermeasure: Template-Enhanced DPA

![DPA of the masked s-box output bit of the test device](image1)

![DPA of the s-box output mask bit of the target device](image2)

Figure 5.5.: Smart card B: DPA of the masked s-box output bit using the test device and DPA of the mask bit using the target device (both with wrong hypothesis).

If the joint distributions in the templates for the zero and one bit are equal. Let \( p_0 \) and \( p_1 \) denote two \( L \)-dimensional random variables with multivariate Gaussian probability distributions corresponding to the target bit being equal to 0 and 1, respectively. Let \( (T_0, C_0) \) and \( (T_1, C_1) \) denote the corresponding ideal, error-free templates (derived if the RNG bias of the test implementation was \( \nu = 1 \)). Then, the templates can be expressed as

\[
T_0 = \begin{pmatrix}
E[p_{0,1}] \\
\vdots \\
E[p_{0,L}]
\end{pmatrix} \\
C_0 = \begin{pmatrix}
COV[p_{0,1}, p_{0,1}] & COV[p_{0,1}, p_{0,2}] & \cdots & COV[p_{0,1}, p_{0,L}] \\
\vdots & \vdots & \ddots & \vdots \\
COV[p_{0,L}, p_{0,1}] & COV[p_{0,L}, p_{0,2}] & \cdots & COV[p_{0,L}, p_{0,L}]
\end{pmatrix}
\]

and

\[
T_1 = \begin{pmatrix}
E[p_{1,1}] \\
\vdots \\
E[p_{1,L}]
\end{pmatrix} \\
C_1 = \begin{pmatrix}
COV[p_{1,1}, p_{1,1}] & COV[p_{1,1}, p_{1,2}] & \cdots & COV[p_{1,1}, p_{1,L}] \\
\vdots & \vdots & \ddots & \vdots \\
COV[p_{1,L}, p_{1,1}] & COV[p_{1,L}, p_{1,2}] & \cdots & COV[p_{1,L}, p_{1,L}]
\end{pmatrix}
\]
As mentioned in the previous section, a bias $\nu \neq 1$ will introduce a mixing error into the two templates. As a matter of fact, the incorrect assignation of traces to the 0-partition and 1-partition results in bimodal distributed $L$-dimensional random variables $p^*_{0}$ and $p^*_{1}$:

$$
\begin{align*}
p^*_{0,1} &= \nu \cdot p_{0,1} + (1 - \nu) \cdot p_{1,1} \\
&\vdots \\
p^*_{0,L} &= \nu \cdot p_{0,L} + (1 - \nu) \cdot p_{1,L}
\end{align*}
$$

and likewise

$$
\begin{align*}
p^*_{1,1} &= \nu \cdot p_{1,1} + (1 - \nu) \cdot p_{0,1} \\
&\vdots \\
p^*_{1,L} &= \nu \cdot p_{1,L} + (1 - \nu) \cdot p_{0,L}
\end{align*}
$$

Let $(\overrightarrow{T}_0^*, C_0^*)$ and $(\overrightarrow{T}_1^*, C_1^*)$ denote the corresponding erroneous templates for an RNG bias $\nu \neq 1$ which are used to classify the target bit equal to 0 and 1. Let us focus on the template $(\overrightarrow{T}_0^*, C_0^*)$ for now (the derivation of template $(\overrightarrow{T}_1^*, C_1^*)$ is analog). It is straightforward that the vector of means becomes

$$
\overrightarrow{T}_0^* = \nu \cdot \overrightarrow{T}_0 + (1 - \nu) \cdot \overrightarrow{T}_1
$$

(5.10)
Any element of the covariance matrix $C_0^*$, i.e. $COV[p_{0,i}^*, p_{0,j}^*]$, can be expressed as

$$COV[p_{0,i}^*, p_{0,j}^*] = E[p_{0,i}^* \cdot p_{0,j}^*] - E[p_{0,i}^*] \cdot E[p_{0,j}^*] = \nu \cdot E[p_{0,i} \cdot p_{0,j}] + (1 - \nu) \cdot E[p_{1,i} \cdot p_{0,j}] - E[p_{0,i}] \cdot E[p_{0,j}^*]$$

$$= \nu \cdot (COV[p_{0,i}, p_{0,j}] + E[p_{0,i}] \cdot E[p_{0,j}]) + (1 - \nu) \cdot (COV[p_{1,i}, p_{1,j}] + E[p_{1,i}] \cdot E[p_{1,j}]) - E[p_{0,i}] \cdot E[p_{0,j}]$$

$$= \nu \cdot COV[p_{0,i}, p_{0,j}] + (1 - \nu) \cdot COV[p_{1,i}, p_{1,j}] + \nu \cdot (1 - \nu) \cdot (E[p_{0,i}] - E[p_{1,i}]) \cdot (E[p_{0,j}] - E[p_{1,j}])$$

Let $\Delta I$ denote the vector of difference of means, i.e.

$$\Delta I = \bar{T}_0 - \bar{T}_1$$

(5.11)

Then, the covariance matrix $C_0^*$ can be expressed as

$$C_0^* = \nu \cdot C_0 + (1 - \nu) \cdot C_1 + \nu \cdot (1 - \nu) \cdot \Delta I \cdot (\Delta I)^T$$

(5.12)

Similarly, template $(\bar{T}_1, C_1^*)$ can be expressed as

$$\bar{T}_1 = \nu \cdot \bar{T}_1 + (1 - \nu) \cdot \bar{T}_0$$

$$C_1^* = \nu \cdot C_1 + (1 - \nu) \cdot C_0 + \nu \cdot (1 - \nu) \cdot \Delta I \cdot (\Delta I)^T$$

(5.13)

(5.14)

During the hypothesis testing phase, an adversary would use the distorted templates $(\bar{T}_0^*, C_0^*)$ and $(\bar{T}_1^*, C_1^*)$ to classify the target bit from a captured side-channel emanation $s$ with $L$ sample points. As mentioned in Section 5.1, the decision criterion is given by

$$\left(s - \bar{T}_0^*\right)^T (C_0^*)^{-1} \left(s - \bar{T}_0^*\right) - \left(s - \bar{T}_1^*\right)^T (C_1^*)^{-1} \left(s - \bar{T}_1^*\right) > \ln(|C_1^*|) - \ln(|C_0^*|)$$

(5.15)

where a decision is made in favor of template $(\bar{T}_1^*, C_1^*)$, if the above inequality is true, and in favor of $(\bar{T}_0^*, C_0^*)$ otherwise. By assuming $C_0 = C_1 = C^o$, (5.15) can be reduced to the following [Tre68]

$$(\bar{T}_1^* - \bar{T}_0^*)^T (C^o)^{-1} s > \frac{1}{2} \left((\bar{T}_1^*)^T (C^o)^{-1} \bar{T}_1^* - (\bar{T}_0^*)^T (C^o)^{-1} \bar{T}_0^*\right)$$

(5.16)

By using (5.10) and (5.13) along with the symmetry of inverses of covariance matrices to cancel common terms, we can further simplify (5.16) to

$$\left(\Delta I\right)^T (C^o)^{-1} s > \frac{1}{2} \left(\bar{T}_1^T (C^o)^{-1} \bar{T}_1 - \bar{T}_0^T (C^o)^{-1} \bar{T}_0\right)$$

(5.17)

\footnote{In our experiments, this assumption holds well.}
Note that $(\Delta I)^T (C^*)^{-1} s$ is a linear combination of Gaussian variables. As a result, under the hypothesis that the target bit is zero, $(\Delta I)^T (C^*)^{-1} s$ is Gaussian distributed with the following mean and variance

\[
E[(\Delta I)^T (C^*)^{-1} s] = (\Delta I)^T (C^*)^{-1} T_0
\]

\[
V[(\Delta I)^T (C^*)^{-1} s] = (\Delta I)^T (C^*)^{-1} C(C^*)^{-1} \Delta I
\]

Let $Q(x), x \geq 0$ denote the probability of a Gaussian random variable with mean 0 and variance 1 being larger than $x$. Under the hypothesis that the target bit is zero, the probability of error incurred by using the distorted templates is given by

\[
P(\text{error}) = Q\left(\frac{\frac{1}{2}(T_1^T (C^*)^{-1} T_1 - T_0^T (C^*)^{-1} T_0) - (\Delta I)^T (C^*)^{-1} \Delta I}{\sqrt{(\Delta I)^T (C^*)^{-1} C(C^*)^{-1} \Delta I}}\right)
\]

We can express the numerator of $Q(\cdot)$ in the above equation solely in terms of $\Delta I$ by realizing that $T_1^T (C^*)^{-1} T_0$ is one dimensional and therefore it equals to its transpose $T_0^T (C^*)^{-1} T_1$.

\[
\frac{1}{2}(T_1^T (C^*)^{-1} T_1 - T_0^T (C^*)^{-1} T_0) = \frac{1}{2} T_1^T (C^*)^{-1} T_1 + \frac{1}{2} T_0^T (C^*)^{-1} T_0 - \frac{1}{2} T_1^T (C^*)^{-1} T_0 - \frac{1}{2} T_0^T (C^*)^{-1} T_1
\]

\[
= \frac{1}{2} T_1^T (C^*)^{-1} \Delta I - \frac{1}{2} T_0^T (C^*)^{-1} \Delta I
\]

\[
= \frac{1}{2} (\Delta I)^T (C^*)^{-1} \Delta I
\]

Thus, probability of error can be expressed as

\[
P(\text{error}) = Q\left(\frac{\frac{1}{2}(\Delta I)^T (C^*)^{-1} \Delta I}{\sqrt{(\Delta I)^T (C^*)^{-1} C(C^*)^{-1} \Delta I}}\right)
\]

Our task is to prove that the argument of $Q(\cdot)$ in the above equation is independent of $\nu$, and therefore, the probability of error in hypothesis testing phase is independent of the RNG bias. Our strategy is to factorize the numerator and denominator of the argument of $Q(\cdot)$ in (5.21), and show that factors involving $\nu$ cancel each other out. The first step towards this factorization is to obtain an expression for $(C^*)^{-1}$ in terms of $C^{-1}$ by using the matrix inversion lemma. The matrix inversion lemma states that for arbitrary matrices $A, U, C$, and $V$, with the only restriction that inverses of $A$ and $C$ exist and the product $UCV$ and the sum $A + UCV$ are well-defined, the following holds true

\[
(A + UCV)^{-1} = A^{-1} - A^{-1} U(C^{-1} + VA^{-1} U)^{-1} VA^{-1}
\]
5.3 Breaking the Masking Countermeasure: Template-Enhanced DPA

Substituting $A = C, U = \nu(1 - \nu)\Delta I, C = 1, $ and $V = (\Delta I)^T$, we obtain

$$(C^*)^{-1} = (C + \nu(1 - \nu)\Delta I \cdot 1 \cdot (\Delta I)^T)^{-1}$$

$$= C^{-1} - \nu(1 - \nu)C^{-1}\Delta I \left(1 + \nu(1 - \nu)\sum_{\beta} (\Delta I)^T C^{-1} \Delta I \right)(\Delta I)^T C^{-1}$$

Let $\beta = (\Delta I)^T C^{-1} \Delta I$. Since $\beta$ is a one dimensional quantity, it can be factored out to obtain

$$(C^*)^{-1} = C^{-1} - \nu(1 - \nu)(1 + \nu(1 - \nu)\beta)\Delta I (\Delta I)^T C^{-1}$$

Now we are ready to factorize the numerator.

$$(\Delta I)^T(C^*)^{-1}\Delta I = (\Delta I)^T C^{-1} \Delta I - \nu(1 - \nu)(1 + \nu(1 - \nu)\beta)(\Delta I)^T C^{-1} \Delta I (\Delta I)^T C^{-1} \Delta I$$

$$= \beta(1 - \nu\beta(1 - \nu)(1 + \nu(1 - \nu)\beta))$$

Similarly, to factorize the denominator, we perform the following steps.

$$(\Delta I)^T(C^*)^{-1}C(C^*)^{-1}\Delta I$$

$$= (\Delta I)^T(C^*)^{-1}\left(I - \nu(1 - \nu)(1 + \nu(1 - \nu)\beta)\Delta I (\Delta I)^T C^{-1}\right)\Delta I$$

$$= ((\Delta I)^T(C^*)^{-1}\Delta I)\left(1 - \nu\beta(1 - \nu)(1 + \nu(1 - \nu)\beta)\right)$$

$$= \beta\left(1 - \nu\beta(1 - \nu)(1 + \nu(1 - \nu)\beta)\right)^2$$

Using (5.24) and (5.25), the numerator and denominator of (5.21) can be simplified to give the following expression for probability of error

$$P(\text{error}) = Q\left(\frac{1}{2}\sqrt{\beta}\right)$$

Note that since $C^{-1}$ is a positive definite matrix, $\beta > 0$. Furthermore, $\beta$ only depends on the statistics of emanations, i.e. $(\Delta I)^T, C^{-1}$ and $\Delta I$. In particular, it does not depend on $\nu$. 
6. Vulnerabilities of Masked AES Hardware Implementations

In this chapter we discuss the security risks of masked AES hardware implementations which -in theory- are supposed to resist side channel attacks. We begin with a brief overview of masking techniques at the gate level and their resitivity to side channel attacks in Section 6.1. Next, in Section 6.2 we present the results of three different attacks (i.e. zero-offset DPA, toggle-count DPA and zero-input DPA) against a masked AES hardware implementation. This discussion leads to the conclusion that glitches in masked circuits pose the biggest threat to masked hardware implementations. Motivated by this fact, we pinpoint which parts of masked AES S-boxes prevent the propagation of glitches for certain inputs and thus lead to side channel leakage in Section 6.3. The analysis reveals that the propagation of glitches is cut off by the switching characteristics of XOR gates in masked multipliers. Masked multipliers are the basic building blocks of most recent proposals for masked AES S-boxes. We subsequently show in Section 6.4 that the side channel leakage of masked multipliers can be eliminated by enforcing timing constraints for the XOR gates in each $GF(2^n)$ multiplier of an AES S-box. We also briefly present two approaches on how these timing constraints can be realized in practice. Parts of this work were submitted to the Cryptographic Hardware and Embedded Systems (CHES) 2006 conference in [MS06].

6.1. Previous Work

As discussed in Section 4.1, one approach to secure software and hardware implementations of AES against power analysis attacks is to mask intermediate key-dependent values that occur during the execution of the algorithm. Masking schemes for AES have been presented in [AG01], [TSG02], [GT02], [MA04], [BGK04], and [OMPR05]. The first two of these schemes have turned out to be susceptible to so-called zero-value attacks [GT02] and the second one is even susceptible to standard DPA attacks [ABG04]. The third scheme is quite complex to implement and there is no published implementation of this approach so far. The last three schemes are provably secure against DPA
attacks and can also be efficiently implemented in hardware. This is why these schemes are the most commonly used schemes to secure implementations of AES in hardware.

Unfortunately, in 2005 several publications showed that even provably secure masking schemes can be broken in practice, if they are implemented in standard CMOS logic. The reason for this is that in CMOS circuits a lot of unintended switching activities occur. These unintended switching activities are often referred to as dynamic hazards or glitches. Mangard et al. analyzed the effect of glitches in masked circuits and their impact on side channel attacks in [MPG05]. A similar analysis has also been presented in [SSI04]. A technique to model the effect of glitches on the side channel resistance of circuits has been published in [SSI05]. The fact that glitches can indeed make circuits susceptible to DPA attacks in practice was finally shown in [MPO05], however, no justification was given why the attack actually worked. These publications made clear that glitches cannot be ignored when implementing masking schemes in hardware. However, the existing articles demonstrated that implementations of masking schemes leak side channel information, but they did not explain which parts of masked circuits account for the leakage.

We will explain why masked multipliers, which are for example the basic building block of masked S-boxes such as [MA04], [OMPR05] and [BGK04], are responsible for side channel leakage. In fact, we will show that the switching characteristics of the XOR summers at the output stage of these multipliers account for the side channel leakage. However, in Section 6.2 we first briefly discuss different DPA attacks on masked AES hardware implementations that have been published recently. In particular, we compare the toggle-count DPA attack presented in [MPO05] with the zero-offset DPA attack presented in [WW04]. Both attacks are performed against the same masked AES hardware implementation according to [OMPR05]. As a result, it turns out that the toggle-count attack is significantly more effective. In fact, we are even able to show that a much simpler power model than the one used in the toggle-count attack results in a successful attack (we call it zero-input attack), as well. The side channel leakage that is caused by glitches hence poses the biggest problem of masked AES hardware implementations in practice.

Motivated by this fact we analyze which parts of the AES S-box actually cause the side channel leakage in Section 6.3. As already pointed out, this analysis leads to the conclusion that the XOR gates within the masked multipliers of the AES S-box account for the side channel leakage. This insight is used in Sect. 6.4 to present new approaches in order to securely implement masking schemes in hardware.
6.2 Attacks on Masked AES Hardware Implementations

In this section we discuss the results of three DPA attacks against a masked AES hardware implementation. The device under attack is an AES ASIC that is based on the masking scheme proposed by Oswald et al. in [OMPR05]. The design is based on the 32-bit architecture shown in Figure 6.1. In this architecture, a column of the AES state is processed within a single cycle. Hence, the computation of one AES round takes four clock cycles, and a complete AES encryption takes 40 clock cycles. All of our following attacks are based on a set of 1,000,000 power traces which we collected from the masked AES chip. The traces were measured at 1 GS/s using a differential probe.

The first attack we discuss in Section 6.2.1 is the zero-offset DPA (ZODPA) as proposed in [WW04]. This attack requires that masks and masked data of the attacked device leak simultaneously. It uses squaring as a preprocessing step. Subsequently, we discuss a DPA attack based on a toggle-count power model of a masked S-box of our chip in Section 6.2.2. This attack has been performed in the same way as it was proposed by Mangard et al. in [MPO05]. Finally, in Section 6.2.3 we present a simplification of this attack, which we refer to as zero-input DPA. This attack is based on the fact that the power consumption of our masked AES S-box implementation has a significant minimum, if the mask and the masked input are equal.
6.2.1. Zero-Offset DPA

The idea of zero-offset DPA was originally proposed by Waddle and Wagner in [WW04]. It represents a special case of second order DPA [Mes00b, JPS05, OMHT06, SP06], because it presumes that a mask and the corresponding masked data leak simultaneously in a power trace. While this scenario is unlikely to happen in masked software implementations, it commonly occurs in masked hardware implementations. In particular, it also occurs in our attacked AES ASIC and hence a zero-offset DPA should theoretically be possible. Zero-offset DPA uses squaring of power traces as a preprocessing step. Let the power consumption at time $t_0$ of the target hardware be

$$P(t_0) = \epsilon \cdot (W(M) + W(Y)) + N$$

(6.1)

$$= \epsilon \cdot \left( \sum_{i=0}^{n-1} M[i] + \sum_{i=0}^{n-1} Y[i] \right) + N$$

(6.2)

where $M$ represents a random $n$-bit mask, $Y = X \oplus M$ represents $n$ bits of key-dependent data masked by $M$, and $N$ represents additive Gaussian noise with mean $\mu$ and variance $\sigma^2$. When squaring this power signal, it becomes clear that a zero-offset DPA is essentially equivalent to a second order DPA (see Chapter 4.4), because the term $W(M) \cdot W(Y)$ occurs in the equation and thus contributes to the leakage.

$$P^2(t_0) = \epsilon^2 \cdot (W(M) + W(Y))^2 + 2 \cdot \epsilon \cdot (W(M) + W(Y)) \cdot N + N^2$$

(6.3)

$$= \epsilon^2 \cdot (W^2(M) + 2 \cdot W(M) \cdot W(Y) + W^2(M))$$

$$+ 2 \cdot \epsilon \cdot (W(M) + W(Y)) \cdot N + N^2$$

(6.4)

$$= \epsilon^2 \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right)^2 + 2\epsilon \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right) \cdot N + N^2$$

(6.5)

The correlation coefficient of the squared power signal and the hypothesized Hamming weight $W(X) = \sum_{i=0}^{n-1} X[i]$ is

$$\rho \left( P^2(t_0), W(X) \right) = \frac{COV[P^2(t_0), W(X)]}{\sqrt{V[P^2(t_0)]} \cdot \sqrt{V[W(X)]}}$$

(6.6)

$$= \frac{E[P^2(t_0) \cdot W(X)] - E[P^2(t_0)] \cdot E[W(X)]}{\sqrt{V[P^2(t_0)]} \cdot \sqrt{V[W(X)]}}$$

(6.7)

Solving all occurring expectation values in the nominator results in

$$E[\epsilon^2 \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right)^2 \cdot \left( \sum_{i=0}^{n-1} X[i] \right)] = \epsilon^2 \cdot \frac{1}{4} \cdot (2n^3 + n^2 - n)$$

(6.8)

$$E[2 \cdot \epsilon \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right) \cdot N \cdot \left( \sum_{i=0}^{n-1} X[i] \right)] = \epsilon \cdot \mu \cdot n^2$$

(6.9)
\[
E[N^2 \cdot \left( \sum_{i=0}^{n-1} X[i] \right)] = (\sigma^2 + \mu^2) \cdot \frac{n}{2} \tag{6.10}
\]

\[
E[e^2 \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right)^2] = e^2 \cdot \frac{1}{2} \cdot (2n^2 + n) \tag{6.11}
\]

\[
E[2 \cdot \epsilon \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right) \cdot N] = 2 \cdot \epsilon \cdot n \cdot \mu \tag{6.12}
\]

\[
E[N^2] = (\sigma^2 + \mu^2) \tag{6.13}
\]

\[
E[W(X)] = E[\left( \sum_{i=0}^{n-1} X[i] \right)] = \frac{n}{2} \tag{6.14}
\]

\[
\Rightarrow \text{COV}[P^2(t_0), W(X)] = -n \cdot \epsilon^2 \cdot \frac{1}{4} \tag{6.15}
\]

The variance of a sum of \( l \) random variables \( x_i \) is defined as

\[
V[\sum_{i=1}^{l} x_i] = \sum_{i=1}^{l} \sum_{j=1}^{l} \text{COV}[x_i, x_j] \tag{6.16}
\]

Hence, solving for the variances in the denominator of the correlation coefficient results in

\[
V[e^2 \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right)] = e^4 \cdot (2n^3 + \frac{n^2}{2} - \frac{n}{4}) \tag{6.17}
\]

\[
V[2 \cdot \epsilon \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right) \cdot N] = 4 \cdot \epsilon^2 \cdot \left( \frac{n}{2} \cdot (\mu^2 + \sigma^2) + n^2 \cdot \sigma^2 \right) \tag{6.18}
\]

\[
V[N^2] = E[N^4] - (E[N^2])^2 = 2 \cdot \sigma^2 \cdot (2 \cdot \mu^2 + \sigma^2) \tag{6.19}
\]

\[
\text{COV}[e^2 \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right)^2, 2 \cdot \epsilon \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right) \cdot N] = 2 \cdot \epsilon^3 \cdot \mu \cdot n^2 \tag{6.20}
\]

\[
\text{COV}[e^2 \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right)^2, N] = 0 \tag{6.21}
\]

\[
\text{COV}[N, 2 \cdot \epsilon \cdot \left( \sum_{i=0}^{n-1} Y[i] + \sum_{i=0}^{n-1} M[i] \right) \cdot N] = 4 \epsilon \cdot n \cdot \sigma^2 \cdot \mu \tag{6.22}
\]

\[
\Rightarrow V[P^2(t_0)] = e^4(2n^3 + \frac{n^2}{2} - \frac{n}{4}) + 4\epsilon^2 \mu n^2 + 4\epsilon^2(\frac{n}{2}(\mu^2 + \sigma^2) + n^2 \sigma^2) + 8\epsilon n \sigma^2 \mu + 2\sigma^2(2\mu^2 + \sigma^2) \tag{6.23}
\]
\begin{equation}
V[W(X)] = V\left(\sum_{i=0}^{n-1} X[i]\right) = \frac{n}{4}
\end{equation}

\begin{equation}
\Rightarrow \rho(P^2(t_0), W(X)) = \frac{-\frac{1}{2} \cdot \sqrt{n} \cdot \epsilon^2}{(\epsilon^4(2n^3 + \frac{n^2}{2} - \frac{n}{4}) + 4\epsilon^3 \mu n^2 + 4\epsilon^2(\frac{n}{2}(\mu^2 + \sigma^2) + n^2\sigma^2) + ...} \\
... + 8\epsilon n\sigma^2\mu + 2\sigma^2(2\mu^2 + \sigma^2))^0.5
\end{equation}

Under the condition \(n = 8\), the correlation coefficient of the squared power signal \(P^2(t_0)\) and the correctly hypothesized Hamming weight of the unmasked, key-dependent data byte \(X = Y \oplus M\) is

\begin{equation}
\rho(P^2(t_0), W(X)) = \frac{-\epsilon^2}{\sqrt{527\epsilon^4 + 128\mu \epsilon^3 + (136\sigma^2 + 8\mu^2)\epsilon^2 + 32\sigma^2 \epsilon \mu + 2\sigma^2 \mu^2 + \sigma^4}}
\end{equation}

The architecture shown in Figure 6.1 stores 128 bits of masked data in the AES state register and the corresponding 128 mask bits in the mask state register. Moreover, the architecture is designed in such a way that a 32-bit column in the AES state register and in the mask state register are processed in a single clock cycle. Hence, we may assess that additive noise is solely caused by the 24 + 24 = 48 remaining bits which are processed simultaneously during a clock cycle, i.e. the noise has mean \(\mu = \frac{48}{2}\epsilon = 24\epsilon\) and variance \(\sigma^2 = \frac{48}{4}\epsilon^2 = 12\epsilon^2\). Using equation 6.26 we obtain an estimation of the correlation coefficient.

\begin{equation}
\hat{\rho}(P^2(t_0), W(X)) = 0.0055
\end{equation}

As discussed in Chapter 2.6 it is possible to assess the number of required measurements with the help of Fisher’s Z-transformation

\begin{equation}
\hat{N} = 3 + 8 \cdot \left(\frac{Z_\alpha}{\ln \left(\frac{1+\hat{\rho}}{1-\hat{\rho}}\right)}\right)^2 \approx 5.29 \cdot 10^5
\end{equation}

with the confidence interval \(Z_\alpha = \Phi^{-1}(0.9999) = 3.789\). One possibility to perform a zero-offset DPA against the architecture shown in Figure 6.1 is to predict the number of bits that switch in the AES state register after round one. In an unmasked implementation the first byte of the AES state register prior to round one would be equal to \(S^0_{(0,0)} = X_{(0,0)} \oplus K_{(0,0)}\). After round one has been processed the first byte of the AES state register would be equal to

\begin{equation}
S^1_{(0,0)} = \{x\} \cdot S(X_{(0,0)} \oplus K_{(0,0)}) \oplus \{x + 1\} \cdot S(X_{(1,1)} \oplus K_{(1,1)}) \oplus \\
S(X_{(2,2)} \oplus K_{(2,2)}) \oplus S(X_{(3,3)} \oplus K_{(3,3)}) \oplus \\
K_{(0,0)} \oplus S(K_{(1,3)}) \oplus 1
\end{equation}
6.2 Attacks on Masked AES Hardware Implementations

In a zero-offset attack the adversary may try to compute the correlation coefficient of the predicted Hamming distance \( W(S_{0,0}^0 \oplus S_{0,0}^1) \) and the squared power traces. Note that the predicted Hamming distances depend on a 40-bit key hypothesis, which may not be feasible in a practical attack. As an alternative, in a chosen plaintext attack scenario the adversary may keep plaintext bytes \( X_{(1,1)} \), \( X_{(2,2)} \) and \( X_{(3,3)} \) constant for all encryptions and only vary plaintext byte \( X_{(0,0)} \). In this case the first byte of the AES state register after round one reduces to \( S_{0,0}^1 = \{ x \} \cdot S(X_{(0,0)} \oplus K_{(0,0)}) \oplus c \) with \( c = constant \) for all traces. Thus, when predicting the Hamming distance

\[
W(S_{0,0}^0 \oplus S_{0,0}^1) = W((X_{(0,0)} \oplus K_{(0,0)}) \oplus \{ x \} \cdot S(X_{(0,0)} \oplus K_{(0,0)}) \oplus c)) \tag{6.29}
\]

the hypothesis space is reduced to 16 bits, which is feasible in practical attacks. As assessed in equation 6.28 we expected that the minimum complexity of the zero-offset DPA is approximately around 500,000. However, even with 1,000,000 measurements we were not able to perform a successful zero-offset DPA against the masked AES implementation. One reason for this failure could be our unrealistic assumption that the power consumption is only caused by the switchings of the AES state and mask state flip flop registers. As a matter of fact, remaining parts of the circuit, e.g. the S-box and MixColumns circuits, also contribute to the overall power consumption and significantly increase additive background noise. Hence, we must conclude that while a zero offset DPA attack may work in theory, we were not able to successfully apply it against our masked AES implementation with 1,000,000 measurements. In this attack we computed a correlation coefficient of \( r = 0.00268 \). Please note that, as discussed in Chapter 2.6, the standard deviation of estimated correlation coefficients after 1,000,000 measurements is approximately \( \sigma \approx \frac{1}{\sqrt{10^6}} = 0.001 \). Hence, the estimated correlation coefficient \( r = 0.00268 \) is insignificant.

6.2.2. Toggle-Count DPA

In conventional CMOS circuits, signal lines typically toggle several times during a clock cycle. In [MPG05] it was shown that the total number of signal toggles in masked non-linear gates, such as masked AND or masked OR gates, is correlated with the states of the unmasked input and output signals. This fact was exploited in simulated DPA attacks. A similar approach was used in [MPO05] to break a fabricated masked AES chip. In this attack Mangard et al. used a back-annotated netlist of the attacked device in order to derive a toggle-count model of the masked AES S-boxes used in the implementation. Then, they demonstrated that the derived number of toggles\(^1\) and measured power traces of the masked AES chip are correlated, if the secret key is predicted correctly.

\(^1\)Note that this attack assumes that each signal toggle has an equal contribution to the power consumption. This condition is typically not met in real life. Nevertheless, the model is usually sufficient to mount successful DPA attacks on masked implementations.
In order to reproduce these results, we performed a similar attack against our masked AES ASIC implementation. We first simulated our design in order to determine the average number of toggles that occur in the masked AES S-box for different data inputs\(^2\). The power model of our S-box is shown in Figure 6.2. In this figure, the number of toggles of our masked S-box is shown for all possible 256 S-box inputs. Please note that there occurs a distinct minimum for S-box input 0, i.e. the case when mask and masked data are equal.

We used the power model shown in Figure 6.2 to mount a DPA attack on our masked AES chip. We correlated the measured power traces of our masked AES implementation with toggle count hypotheses based on the power model. In this attack, we obtained a correlation coefficient of \( r = 0.04 \) for the correct key hypothesis using 1,000,000 mea-

\(^2\)We denote the term \textit{data input} as the input mask x-or the masked input of the S-box.
6.2 Attacks on Masked AES Hardware Implementations

Figure 6.4.: Correlation coefficients of a zero-input DPA against the masked AES ASIC with 30,000 measurements. The correct key hypothesis (225) is clearly distinguishable from the false correlation coefficients.

measurements. Approximately only 15,000 measurements were necessary to distinguish this correlation coefficient from the 255 correlation coefficients of incorrect key hypotheses. The correlation coefficients for an attack based on 15,000 measurements are shown in Figure 6.3.

6.2.3. Zero-Input DPA

As shown in Figure 6.2, the simulated masked AES S-box has a significant power consumption minimum, if the S-box input \( x = x_m \oplus m_x = 0 \). This significant minimum suggests that it should also be possible to perform DPA attacks which just exploit this characteristic. Hence, we adapted our power model of the S-box to the following much simpler model \( P(x) \).

\[
P(x) = 0 \quad \text{if} \quad x = 0 \\
= 1 \quad \text{if} \quad x \neq 0
\]

Using this generic zero-input power model instead of the toggle-count model we repeated our attack based on the same set of power traces. We obtained a correlation coefficient of \( r = 0.022 \) for the correct key hypothesis. About 30,000 measurements were necessary to clearly distinguish this correlation coefficient from the ones of false key hypotheses. Figure 6.4 shows the result of an attack based on 30,000 measurements.

As expected the number of measurements that are required for a zero-input DPA is greater compared to the attack based on the more precise toggle-count model. However, the attack is still feasible and it is much more effective than zero-offset DPA. The biggest advantage of the zero-input DPA over the two other attacks we have discussed, is the
<table>
<thead>
<tr>
<th>Type of Attack</th>
<th>Number of measurements</th>
<th>Correlation coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero-offset DPA</td>
<td>1,000,000</td>
<td>0.00268</td>
</tr>
<tr>
<td>toggle count DPA</td>
<td>1,000,000</td>
<td>0.03916</td>
</tr>
<tr>
<td>toggle count DPA</td>
<td>15,000</td>
<td>0.03830</td>
</tr>
<tr>
<td>zero-input DPA</td>
<td>1,000,000</td>
<td>0.02245</td>
</tr>
<tr>
<td>zero-input DPA</td>
<td>30,000</td>
<td>0.02598</td>
</tr>
</tbody>
</table>

Table 6.1: Results of various DPA attacks against a masked hardware implementation of AES. All attacks except for the zero-offset DPA were successful.

The fact that zero-input DPA does not require detailed knowledge about the attacked device yet it is still very successful. It exploits the fact that the power consumption of the masked S-box implementation has a significant minimum for the input value zero. In the following section, we discuss why implementations of masked S-boxes actually leak side channel information and we pinpoint at which parts of the circuit the side channel leakage is caused.

The results of all attacks are shown in Table 6.1. As a result, while a zero-offset DPA was not possible with 1,000,000 measurements, both the toggle count and zero-input DPA succeeded with far less measurements. In the following Section 6.3 we explain why these two attacks work so well in practice.

6.3. Pinpointing the side channel Leakage of Masked S-boxes

The masked AES S-box implementation we have attacked in the previous section is based on composite field arithmetic. In fact, most recent proposals for masked AES S-boxes (see [MA04], [OMPR05], and [BGK04]) are based on this approach. Masked AES S-boxes of this kind essentially consist of an affine transformation, isomorphic mappings, adders and multipliers. All these elements except for the multipliers are linear and hence it is easy to mask them additively. An additive masking of a linear operation can be done by simply performing the operation separately for the masked data and the mask.

In hardware, masked linear operations are usually implemented by two completely separate circuits. One circuit performs the linear operation for the masked data and one circuit performs the linear operation for the corresponding mask. There is no shared signal line between these two circuits. Therefore, the power consumption $P_1$ of the first circuit exclusively depends on the masked data and the power consumption $P_2$ of the second circuit exclusively depends on the mask. According to the definition of additive
6.3 Pinpointing the side channel Leakage of Masked S-boxes

masking [AG01], the masked data and the mask are both statistically independent from the corresponding unmasked data. Hence, the power consumptions $P_1$ and $P_2$ are also pairwise statistically independent from the unmasked data.

In practice this means that an attacker who does not know the value of the mask can not perform a successful DPA attack on the power consumption of either of these two circuits. An attacker can only make hypotheses regarding unmasked intermediate values of the performed cryptographic algorithm. In this article, we denote the set of all unmasked intermediate values of the attacked algorithm as $\mathcal{H}$. Our previous argumentation hence formally means that $\rho(H, P_1)$ and $\rho(H, P_2)$ are both 0 for all $H \in \mathcal{H}$. This also implies that the total power consumption is uncorrelated to all intermediate values, i.e. $\rho(H, P_1 + P_2) = 0 \forall H \in \mathcal{H}$. Throughout this article, we use the common assumption that the total power consumption of a circuit is the sum of the power consumption of its components. Using this assumption, it is clear that the linear elements of a masked S-box do not account for the side channel leakage we have observed in Sect. 6.2. The side channel leakage can only be caused by the non-linear elements, i.e. the S-boxes which combine masks and masked data.

In general, there exist several approaches to implement a S-box based on masked multipliers. There is a very common approach of a masked $GF(2^n)$ multiplier which is shown in Figure 6.5. The multiplier takes two masked $n$-bit inputs $a_m$ and $b_m$ that are masked with $m_a$ and $m_b$, respectively. The $n$-bit output $q_m$ is the product of the corresponding unmasked values $a$ and $b$ masked by $m_q$.

![Image of a masked multiplier](image)

Figure 6.5.: Common architecture of a masked multiplier.

The masked multiplier consists of four unmasked multipliers that calculate the intermediate values $i_1 \ldots i_4$. These intermediate values are then summed by $4 \cdot n$ XOR gates. A
masked multiplier of this kind was proposed as a masked AND gate \((n = 1)\) by Trichina et al. in [Tri03]. Furthermore, this architecture is also used in various masked S-boxes published in [MA04], [BGK04], and [OMPR05].

We start our analysis by first looking at a masked AND gate \((n = 1)\). Subsequently, we look at multipliers in \(GF(2^2)\) and \(GF(2^4)\). Finally, we look at the side channel leakage of masked S-boxes as a whole that contain several such masked multipliers.

### 6.3.1. Masked AND Gate

Glitches and masked AND gates that are based on the architecture shown in Figure 6.5 were analyzed in [MPG05]. This analysis has revealed that such gates indeed leak side channel information. We now look more closely at this fact in order to pinpoint the exact cause of the side channel leakage. For this purpose we implemented a masked AND gate based on the architecture shown in Figure 6.5. Then, we simulated the back-annotated netlist of this gate for all possible input transitions. There are five input signals and hence there are \(2^{10}\) possible input transitions\(^3\). For each of these \(2^{10}\) simulations we counted the number of glitches that occurred on each signal line in the circuit. Let the numbers of transitions be \(T(a_m), T(b_m), T(m_a), T(m_b), T(m_q), T(q_m), \) and \(T(i_1) \ldots T(i_7)\).

In order to analyze which signal lines account for the side channel leakage of the gate, we calculated the correlation coefficients between these signal toggle count numbers on the one hand and the unmasked, and thus predictable, values \(a, b\) and \(q\) on the other hand. Due to the masking \(T(a_m), T(b_m), T(m_a), T(m_b), \) and \(T(m_q)\) do not leak side channel information. Furthermore, it turns out that also \(\rho(T(i_j), a) = 0, \rho(T(i_j), b) = 0\) and \(\rho(T(i_j), q) = 0\) for \(j = 1 \ldots 4\). This result is actually not surprising. The four multipliers (the four AND gates in case of \(n = 1\)) never recombinates a masked value and a corresponding mask. For example, there is no multiplier that takes \(a_m\) and \(m_a\) as input. Each pair of inputs of the multipliers is statistically independent of \(a, b\) and \(q\). Therefore, also the power consumption of the multipliers and their outputs are independent of \(a, b\) and \(q\). The side channel leakage can hence only be caused by the switching activities of the remaining signal lines, i.e. at the inputs and outputs of the XOR gates.

At first sight this might seem counter-intuitive. The number of transitions that occur at the output of an XOR gate intuitively corresponds to the sum of transitions that occur at the inputs of the gate. Each transition at the input of an XOR gate should exactly lead to one output transition. The number of input transitions does not leak side channel information and hence also the number of output transitions should not. In reality this reasoning is unfortunately wrong. It is true that an XOR gate usually switches its output each time an input signal switches. However, the gate does not switch its output, if both input signals switch simultaneously or within a short period.

\(^3\)In our simulations all input signals entered the masked AND gate at the same time.
of time (this temporal threshold depends on the transistor technology used). In this
case, the input transitions are “absorbed” by the XOR gate and not further propagated.
Exactly this effect accounts for the side channel leakage of the masked AND gate. Our
simulations have shown that the number of absorbed transitions is indeed correlated to
the states of the unmasked input \( a, b \) and the output \( q \). This means that the arrival
times of the input signals at the XOR gates depend on the unmasked values. It is the
joint distribution of the arrival times of the signals \( i_1 \ldots i_4 \) that causes the side channel
leakage of the gate. The arrival times are different for different unmasked values and
hence a different number of transitions is absorbed. This in turn leads to a different
power consumption.

It is important to point out that it is exclusively this effect that accounts for the side
channel leakage of the masked AND gate. If each XOR gate would switch its output as
often as its inputs switch, the gate would be secure. This is a consequence of the fact
that \( T(i_1) \ldots T(i_4) \) are uncorrelated to \( a, b \) and \( q \). Based on this insight it is actually
possible to develop secure masked circuits in a new way. We elaborate further on this
issue in Section 6.4.

6.3.2. Masked Multipliers for \( GF(2^2) \) and \( GF(2^4) \)

In order to confirm the conclusion we made in the previous section regarding the masked
AND gate, we also implemented masked multipliers for the finite fields \( GF(2^2) \) and
\( GF(2^4) \). Multipliers of this kind are the basic building block in the masked composite
field-based AES S-boxes of [MA04], [BGK04], and [OMPR05]. As in the case of the
masked AND gates, we performed different simulations based on back-annotated netlists
of these multipliers.

First, we confirmed that \( T(i_1) \ldots T(i_4) \) are indeed independent of \( a, b \) and \( q \). This
analysis was actually just done for sake of completeness. From a theoretical point of view
it is clear that the power consumption of the four multipliers shown in Figure 6.5 must be
independent of the unmasked values. As mentioned above, the inputs of each multiplier
are completely statistically independent from the unmasked input. This reasoning is
independent of the bit width of the multipliers.

In the second step, we analyzed the switching characteristics of the XOR gates. Our
simulations confirmed that the number of absorbed transitions is correlated with the
unmasked values of \( a, b \) and \( q \)—exactly as in the case of the masked AND gate. The
side channel leakage of all masked multipliers that are based on the architecture shown
in Figure 6.5 is obviously caused by the same effect. However, unfortunately it is not
de possible to make a general statement regarding the leakage of the various masked multi-
pliers. The fact how many transitions are absorbed by the XOR gates depends on many
implementation details. The arrival times of the signals at the XOR gates strongly de-
pend on the placement and routing of the circuit. Moreover, the used CMOS library has
a strong impact, as well. The library affects the timing of the input signals and it also
determines how big the delay between two input transitions of an XOR gate has to be
in order propagate.

Based on our experiments, we can make one general statement. We have implemented
several masked multipliers and we have also placed and routed them several times. In
all cases, we have observed side channel leakage. In order to prevent that the XOR gates
absorb transitions, it is therefore necessary to explicitly take care of this issue during
the design process (see Sect. 6.4).

6.3.3. Masked AES S-boxes

Masked AES S-boxes as they are presented in [MA04, OMPR05, BGK04] contain
several masked multipliers. As pointed out before, these multipliers account for the
side channel leakage of the S-boxes. We now analyze two concrete implementations of
masked AES S-boxes in order to check how the side channel leakage of the multipliers
affects other components of the S-boxes. First, we analyze an implementation of the
AES S-box proposed in [OMPR05] and then we look at an implementation of [MA04].

Masked S-box of Oswald et al.

The first step of our analysis was to generate a back-annotated netlist of the masked AES
S-box described in [OMPR05]. Subsequently, we have simulated this netlist for 200,000
randomly selected input transitions. During these simulations, we have counted the
number of transitions that occur on each of the internal signal lines of the S-box. Based
on these numbers it was possible to determine which signal lines cause the most side
channel leakage.

As expected, all linear operations that are performed at the beginning of the S-box
do not leak any information. The transitions that occur on the corresponding signal
lines are independent of the unmasked S-box input. The first leakage within the S-box
occurs in the first masked multiplier. The XOR gates of this multiplier absorb a different
number of transitions for different data inputs. The total number of transitions which
is related to the power consumption of the masked multiplier is therefore correlated to
the state of the unmasked S-box input. Please note that the unmasked S-box input of
AES in the first round can usually be predicted by an adversary, because it depends on
a known plaintext byte and an 8-bit key hypothesis.

The fact that the switching activity of an XOR gate is correlated to the unmasked S-
box input has severe consequences for all components that use the output of the XOR
gate as input. The switching activity of all these subsequent components typically also
6.3 Pinpointing the side channel Leakage of Masked S-boxes

becomes correlated to the unmasked S-box input\(^4\). This holds true for linear and non-linear components. Therefore, the switching activities of an XOR gate that occurs the first masked multiplier affects subsequent parts of the S-box and spreads out like an avalanche.

The leakage is additionally amplified by the leakage of other masked multipliers in the S-box. In fact, the leakage continuously grows on its way through the S-box. In case of our S-box implementation of [OMPR05] this leads to the power consumption characteristic we have already shown in Fig. 6.2. A different number of transitions occur for every unmasked S-box input. However, a significant minimum for the number of transitions occurs in the case that the input value is 0, i.e. the masked S-box input and the corresponding mask are equal. We assume that the arrival times of the signals in the masked multipliers are more uniform, if masked input and mask are equal compared to all other cases. Therefore, more transitions are absorbed by the XOR gates and hence less transitions propagate through the subsequent components which are connected to the multipliers. Hence, the power consumption is considerably lower, if masked input and mask are equal.

Masked S-box of Morioka and Akishita

We also analyzed the masked AES S-box proposed by Morioka and Akishita in [MA04]. The architecture of this S-box is based on the unmasked S-box proposed by Satoh et al. in [SMTM01]. As in the case of the masked S-box by Oswald et al. [OMPR05] we first generated a back-annotated netlist of the design. Subsequently, we simulated 200,000 random input transitions and we have the number of transitions for each signal line. Again, we noticed that the total number of transitions in the masked S-box circuit is clearly correlated to the unmasked S-box input. As a matter of fact, we were able to successfully mount a simulated zero-input attack on this masked S-box. The attack only required a few thousand simulated power traces, i.e. simulations of transition counts. This result also confirms our aforementioned claim that a precise power model of a masked S-box implemented in CMOS is not necessarily required to successfully perform a DPA attack.

In order to investigate why the number of toggles has a minimum, if mask and masked input are equal, we evaluated the transition count data of various S-box subcircuits. Then, we performed zero-input attacks against these subcircuits. Exactly as in the case of the masked S-box by Oswald et al. we found out that glitches are absorbed in XOR gates of a masked finite field multiplier. Our analysis also confirmed that the number

\(^4\)There are of course also gates that do not propagate glitches. For example, the output signal of a NAND gate that is connected to a leaking signal on input one and to 0 on input two does not leak any information. However, there are typically sufficient gates connected to a leaking signal that at least some of the gates propagate the leakage created in the XOR gates.
of absorbed transitions is correlated to the unmasked S-box input and that there is a significant power consumption minimum for input 0.

The masked S-box of Morioka and Akishita is highly symmetric with regard to the signal paths of the mask and masked input. This symmetry seems to be the main reason why transitions are absorbed by the XOR gates, if mask and masked input are equal. It is difficult to make a general statement on whether indeed every masked S-box has a significant power consumption minimum, if its input is 0. Many implementation details influence the exact switching characteristics of an S-box. However, based on our observations we assume that most masked S-boxes implemented in standard CMOS logic are vulnerable to zero-input attacks.

6.4. Countermeasures

In the previous section, we analyzed the side channel leakage of masked multipliers that are based on the architecture shown in Figure 6.5. It turned out that the XOR gates which sum the outputs of the four multipliers account for the side channel leakage. These XOR gates can absorb transitions and the number of absorbed transitions is correlated to the unmasked inputs of the masked multiplier.

In Section 6.3, we already pointed out that it is exclusively this absorption that causes the side channel leakage. A masked multiplier is secure against DPA attacks, if no transitions are absorbed by the XOR gates. This means that the number of transitions at the output of an XOR gate must be equal to the total number of transitions occurring at the inputs. A masked multiplier that implements XOR gates in this way is secure. The transitions of the signal lines \( i_1 \ldots i_4 \) are uncorrelated to \( a, b \) and \( q \). If the XOR gates propagate these transitions to the output \( q_m \) without any absorption, the whole multiplier is secure.

In a masked \( GF(2^n) \) multiplier, there are \( 4 \cdot n \) XOR gates that sum the signals \( i_1 \ldots i_4 \) and \( m_q \). An investigation of Figure 6.5 makes clear that the \( n \) XOR gates which sum \( i_4 \) and \( m_q \) are actually not critical. The input signals of these gates depend on mask values only and hence the absorbed number of transitions of these gates cannot depend on \( a, b \) or \( q \). As a consequence, there are actually only \( 3 \cdot n \) XOR gates in a masked multiplier for which an absorption of transitions must be prevented. These are the gates which sum \( i_1, i_2, i_3 \) and \( i_7 \).

Preventing an absorption at these gates means that the inputs of these gates must not arrive simultaneously or within the propagation delay of the XOR gate. This is the timing constraint for the input signals that needs to be fulfilled. In general, timing constraints are quite challenging to fulfill in practice. However, there exist two approaches that can be used to reach this goal.
The first approach is to insert delay elements (e.g., inverter chains) into the paths of the input signals of the XOR gates. A similar approach has actually already been used in [MS03] to reduce the power consumption of an unmasked AES S-box. In case of a masked multiplier, delay elements need be inserted into the lines \( i_1, i_2 \) and \( i_3 \) in such a way that the timing constraints for the XOR gates are always fulfilled. We successfully implemented a secure \( GF(2) \) multiplier based on this approach. Simulations of this multiplier confirmed that the number of transitions of all signal lines in the design are indeed independent of \( a, b \) and \( q \).

![Secure architecture of a masked multiplier using delay chains](image)

\[ q_m = (a \ b) \text{ xor m} \]

Figure 6.6.: Secure architecture of a masked multiplier using delay chains.

However, it is important to point out that it is not always possible to efficiently fulfill the timing constraints of the XOR gates by inserting delay elements. For our masked multiplier, we have assumed that all masked input signals arrive at the same time. However, the arrival times of the inputs of a masked multiplier can vary significantly, if these inputs are not directly connected to flip flops. If the multiplier is part of a long signal path, the approach of inserting delay elements is usually not the best one to fulfill the timing constraints.

An alternative to inserting delay elements is to use enable signals in the circuit. The basic idea of this approach is to generate enable signals by a dedicated circuit that enable the inputs of the critical XOR gates just at the right time. Enable signals of this kind have for example also been used in [SSI04] to control the switching activity of masked gates. Of course, the generation of enable signals requires a certain effort and it increases the design costs.

However, building secure masked circuits is always associated with several disadvantages. The proposal for secure masked gates presented in [FG05] is also associated with timing
constraints that need to be fulfilled when building a masked circuit. One approach for secure masked circuits without timing constraints was presented in [PM05]. However, this approach requires a pre-charging phase and hence the throughput of such implementations is halved compared to standard CMOS circuits. The insight that masked $GF(2^n)$ multipliers can be securely implemented by only fulfilling a timing constraint for $3 \cdot n$ XOR gates definitely enables new competitive designs.
7. Conclusions and Future Work

Side channel attacks represent a continuous struggle for cryptographers, engineers and related industries. Some side channel attacks presume that an adversary has very detailed information about a target implementation, such as power or EM leakage models and precise timing information, and can therefore not be easily performed. Other attacks, such as standard DPA, do not require exact power leakage models nor timing characteristics to be successful. Many countermeasures, e.g. random masking of intermediate, key-dependent variables and desynchronization of power traces, have been proposed in the last years. Unfortunately, countermeasures always implicate performance losses, e.g. a first order masking countermeasure is likely to slow down a software implementation by approximately 50% while it may double the amount of required memory resources. It turns out that a combination of available countermeasures seems to be the best choice.

In this thesis we have investigated both new side channel attacks and corresponding countermeasures, especially the randomized masking of intermediate, key-related data in ciphers. In Chapter 3 we have proposed a new class of side channel attacks which exploits internal collisions in ciphers. We have shown that internal collisions can be detected by analysis of side channel information. We have proposed an internal collision attack against the Data Encryption Standard (DES). We were able to find internal collisions at the output of three adjacent S-boxes in the first round with an average minimum of 140 encryptions which reveals an average of 10.2 key bits. As a proof of concept we have applied the attack on a DES software implementation running on a 8051 microcontroller. In Chapter 3.3, we have exploited partial collisions at the output of the MixColumns transformation used in the Advanced Encryption Standard (AES). We have shown that it is possible to apply the attack in parallel to all four columns in AES. Thus, the entire 128-bit round key can be extracted with only 40 encryptions. As in the case of DES, we have performed the attack against a software assembly implementation of AES and discuss our results. Finally, in Chapters 3.4 and 3.5 we have presented two further applications of internal collision attacks against the block ciphers Serpent and Kasumi, respectively. Due to the birthday paradox all these attacks are extremely efficient and certainly comparable to standard DPA attacks. However, all the target implementations we attacked were unprotected devices. It is relatively easy to thwart internal collision attacks by the use of appropriate countermeasures, such
as randomized masking. Internal collisions are still a very interesting research topic, for example in connection with source code disassembling of security applications. For example, in the future it may be possible to design a reverse engineering application which is able to identify certain program code sequences by side channel analysis. The signal processing techniques used to detect internal collisions and certain code sequences are likely to be very similar.

Since masking play such an important role, we have investigated various masking techniques used to protect AES software implementations against side channel attacks, such as first and higher order DPA and the aforementioned internal collision attacks in Chapter 4. In Chapter 4.3, we have presented an efficient masking scheme for AES software implementations based on inversions in the composite field. Subsequently, in Chapter 4.4, we have discussed the theoretical background of higher order side channel attacks and we have proposed various masking strategies which lead to higher order resistant AES software implementations. We have presented the preformance figures of AES assembly implementations which are resistant against DPA attacks of various orders. Unfortunately, our implementations which resist at least second order DPA attacks turn out to be significantly slower and also require vastly more memory resources due to continuous recomputations of the masked S-boxes. The development of efficient countermeasures which thwart higher order attacks thus represents an open issue which should be further investigated and optimized in the future.

In Chapter 5, we have presented a template-enhanced DPA attack which can defeat the masking countermeasure under certain conditions. The attacks presumes that the adversary has access to a smart card or cryptographic token which is protected by the masking countermeasure, but whose random number generator is defective or has some imperfect bias. We have shown that an adversary can then build templates which allow to break an identical smart card even if its RNG is perfectly functional and has no bias. This attack opens up a back door by giving smart card manufacturers, vendors and developers the possibility to break their own implementations, if they have access to a card, whose RNG is biased, burst or has been intentionally destroyed. As a proof of concept, we have attacked a masked implementation of the DES running on a 6805-based smart card a masked implementation of the AES running on an AVR-based smart card. This attack makes clear that it is crucial to check the entropy and uniformity of an RNG in implementations of cryptographic implementations and only allow valid operation, if these conditions are met.

Finally, in Chapter 6, we have discussed the vulnerabilities of a masked AES hardware implementation realized in CMOS logic. In this context, we have presented the results of three different attacks, i.e. zero-offset DPA, toggle-count DPA and zero-input DPA. Comparison of these attacks has made clear that toggle-count and zero-input DPA require far less measurements than zero-offset DPA. We have shown that these two attacks work so well due to the occurrence of glitches in masked circuits. Motivated by this fact, we have shown which parts of masked AES S-boxes prevent the propagation of glitches
for certain inputs and thus result in a side channel vulnerability. The analysis reveals that the propagation of glitches is significantly influenced by the switching characteristics of XOR gates in masked multipliers. Masked multipliers are the basic building blocks of most recent proposals for masked AES S-boxes based on inversions in the composite field. We have subsequently shown that the side channel leakage of masked multipliers can be prevented by enforcing timing constraints for the XOR gates in each multiplier of an AES S-box. We have also briefly presented two approaches which show how these timing constraints can be realized in practice. It turns out that it is possible to use delay elements, such as inverter chains, to enforce the required timing constraints. However, it is still unclear how difficult it is to generally prevent side channel leakage caused by glitches, because a successful countermeasures which enforces timing conditions has to take many different system and implementation parameters, such as gates delays, layout information, etc., into account.
A. Bibliography


Bibliography


[OS05] E. Oswald and K. Schramm. An Efficient Masking Scheme for AES Software Implementations. In J. Song, T. Kwon, and M. Yung, editors, Workshop on


Johannes Wolkerstorfer, Elisabeth Oswald, and Mario Lamberger. An ASIC implementation of the AES SBoxes. In Bart Preneel, editor, Topics in Cryptology - CT-RSA 2002, The Cryptographer’s Track at the RSA Conference


Curriculum Vitae

Personal Data
Born on March, 7th, 1977 in Essen, Germany
Contact Information: schramm.kai@gmail.com

Secondary Education

1987-1993  Gymnasium, Velbert Langenberg
1993-1994  Palmetto Sr. High School in Miami, Florida, USA
           (School exchange: one-year study program)
1994-1996  Gymnasium, Velbert Langenberg (degree: Abitur)
1996-1999  University of Bochum, Germany
1999-2000  Purdue university, Indiana, USA
           (University exchange: one-year study program)
2000-2002  University of Bochum, Germany
           (degree: M.S. in electrical engineering)

Work & Research Experience

11.2002-present  Research & teaching assistant at the communication security
                 group, Prof. Paar, University of Bochum Germany.
08.2005-10.2005  Internship with the chip card and RFID security group at the
                 Hitachi Central Research Laboratory in Tokyo, Japan
08.2004-10.2004  Internship with the security group at the IBM T.J. Watson
                 Research Center in Hawthorne, NY, USA
10.2001-12.2001  Internship with the chip card & security department
                 of Infineon Technologies, Munich, Germany
09.1994-10.1995  Database programming at Primasoft GmbH

1 as of May 2006
Publications


- K. Schramm, C. Paar, "Higher Order Masking of the AES", RSA conference, Cryptographers' Track, February 13-17, 2006, San Jose, USA.


