

A STUDY OF ADVANCED ENCRYPTION STANDARD SUBSTITUTION-BOX
IMPLEMENTATIONS AND POWER-ANALYSIS RESISTANT DESIGNS



by
Caner Toprak

Submitted to Graduate School of Natural and Applied Sciences
in Partial Fulfillment of the Requirements
for the Degree of Master of Science in
Electrical and Electronics Engineering

Yeditepe University
2019

A STUDY OF ADVANCED ENCRYPTION STANDARD SUBSTITUTION-BOX
IMPLEMENTATIONS AND POWER-ANALYSIS RESISTANT DESIGNS

APPROVED BY:

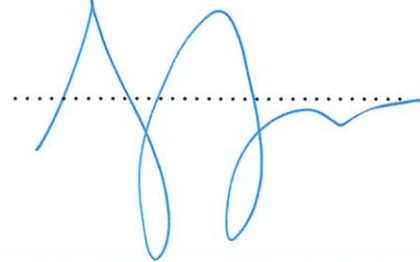
Prof. Dr. Uğur Çilingiroğlu
(Thesis Supervisor)
(Yeditepe University)



Prof. Dr. Fatih Uğurdağ
(Özyeğin University)



Prof. Dr. Sezer Gören Uğurdağ
(Yeditepe University)



DATE OF APPROVAL:/..../2019

ACKNOWLEDGEMENTS

I am thankful to my supervisor Prof. Dr. Uğur Çilingirođlu for introducing me to this idea. Furthermore, I can not thank Elif enough for her help and moral support.



ABSTRACT

A STUDY OF ADVANCED ENCRYPTION STANDARD SUBSTITUTION-BOX IMPLEMENTATIONS AND POWER-ANALYSIS RESISTANT DESIGNS

Today, the subject of symmetric-key algorithms has been extensively studied. Throughout the years, different ones have been developed, scrutinized and standardized. First wide-spread algorithm has been the Data Encryption Standard (DES) which was developed in the 1970s and standardized in 1977 by National Bureau of Standards (NBS). After DES became outdated, National Institute of Standards and Technology (NIST) announced a competition in 2001 to find a new encryption standard that would fulfill the security requirements of its time. At the end of the competition the Rijndael algorithm was selected to be the new symmetric-key algorithm standard by the name of Advanced Encryption Standard (AES). All the explained processes contribute to constant creation and testing of best algorithms mathematically possible. Nonetheless, a chain is as strong as its weakest ring and these algorithms are still being decrypted not because of their algorithmic weaknesses but because of other factors such as user-related or implementational ones. Though, excluding the glaring mistakes, some variables are harder to control. These algorithms are implemented by real devices, and real devices spend time and energy to complete the tasks they are given. These non-idealities mean that these devices leak information of what is going on inside them via electromagnetic emanations, electrical power consumption and timing delays. Therefore, a potential attacker can opt to target the non-idealities of the device rather than the algorithm. This kind of cryptographic attacks are called side-channel attacks. One of the most popular side-channel attacks is the power analysis attack. It relies on the fact that statistical properties of a device's power consumption depend on the operation and the data being operated. For AES, these attacks mostly target the substitution-boxes (S-box) because of their interesting statistical properties. Thus, this project explores different ways to implement S-boxes of AES while attempting to increase their power-analysis resistance in logic gate level.

ÖZET

GELİŞMİŞ ŞİFRELEME STANDARDI İKAME KUTUSU UYGULAMALARI VE GÜÇ ANALİZİNE DAYANIKLI DİZAYNLARIN BİR İNCELEMESİ

Günümüzde simetrik anahtar algoritmaları konusu yoğun olarak incelenmiştir. Yıllar boyunca, farklı algoritmalar geliştirilmiş, incelenmiş ve standartlaştırılmıştır. İlk yaygınlaşan algoritma, 1970'lerde geliştirilen ve 1977'de Ulusal Standartlar Bürosu (NBS) tarafından standartlaştırılan Veri Şifreleme Standardı (DES) olmuştur. DES'in modası geçtikten sonra, Ulusal Standartlar ve Teknoloji Enstitüsü (NIST) 2001 yılında, zamanının güvenlik gereksinimlerini karşılayacak yeni bir şifreleme standardı bulmak için bir yarışma başlattı. Yarışma sonunda Rijndael algoritması, Gelişmiş Şifreleme Standardı (AES) adıyla yeni simetrik anahtar algoritması standardı olarak seçildi. Tarif edilen tüm bu süreçler, sürekli matematiksel olarak mümkün olan en iyi algoritmaların oluşturulmasına ve test edilmesine katkıda bulunuyor. Bununla birlikte, bir zincir en zayıf halkası kadar güçlüdür ve bu algoritmalar algoritmik zayıflıkları nedeniyle olmasa da, kullanıcı veya uygulamayla ilgili diğer faktörler nedeniyle deşifre edilmektedir. Bariz hatalar hariç, bazı değişkenleri kontrol etmek daha zordur. Bu algoritmalar gerçek cihazlar tarafından uygulanır ve gerçek cihazlar verdikleri görevleri tamamlamak için zaman ve enerji harcarlar. Bu idealsizlikler, bu cihazların içlerinde neler olup bittiğini elektromanyetik yaymalar, elektrik enerjisi tüketimi ve zamanlama gecikmeleri vesileleriyle sızdırmasına yol açar. Bu nedenle, potansiyel bir saldırgan, algoritmanın yerine cihazın idealsizliklerini hedeflemeyi seçebilir. Şifrelemeyle ilgili bu tür saldırılara yan kanal saldırıları denir. En popüler yan kanal saldırılarından biri güç analizi saldırısıdır. Bu saldırı, bir cihazın güç tüketimine ait istatistiksel özelliklerinin, işleyişine ve işlenen verilere bağlı olmasına dayanır. AES için bu saldırılar, ilginç istatistiksel özelliklerinden dolayı çoğunlukla ikame kutularını (S-box) hedeflemektedir. Bu nedenle, bu proje AES S-box'ları uygulamak için farklı yollar araştırırken, lojik kapıları seviyesinde güç analizi direncini arttırmaya çalışmaktadır.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS.....	iii
ABSTRACT	iv
ÖZET	v
LIST OF FIGURES	ix
LIST OF TABLES.....	xiii
LIST OF SYMBOLS/ABBREVIATIONS.....	xiv
1. INTRODUCTION.....	1
1.1. BRIEF HISTORY OF MODERN CRYPTOGRAPHY.....	1
1.2. AES S-BOX.....	7
1.3. SIDE-CHANNEL ATTACKS.....	10
1.4. OBJECTIVE OF THE THESIS.....	17
2. STATISTICAL BACKGROUND	18
2.1. NORMAL DISTRIBUTION	18
2.2. ESTIMATION OF PARAMETERS	19
2.3. CONFIDENCE INTERVALS.....	20
2.4. NUMBER OF SAMPLES	21
2.4.1. Number of Samples for ρ	21
2.4.2. Number of Samples for $\rho_0 - \rho_1$	23
2.5. STATISTICAL PROPERTIES OF POWER CONSUMPTION	24
3. POWER ANALYSIS	26
3.1. CORRELATION COEFFICIENT-BASED ATTACK.....	26
3.1.1. Hypothetical Power Consumption Models.....	28
3.2. TEMPLATE-BASED ATTACK.....	29
3.2.1. Choosing the Points of Interest.....	32
3.3. CHOOSING AN INTERMEDIATE VALUE.....	33
4. DESIGNS AGAINST POWER ANALYSIS.....	34

4.1.	TIME DOMAIN COUNTERMEASURES	34
4.2.	AMPLITUDE DOMAIN COUNTERMEASURES.....	35
4.2.1.	Dynamic Differential	36
5.	FINITE FIELDS	49
5.1.	GROUPS.....	49
5.2.	RINGS AND FIELDS	53
5.3.	POLYNOMIALS.....	55
5.4.	MORE ON FIELDS	57
6.	ADVANCED ENCRYPTION STANDARD	63
6.1.	SUBBYTES	63
6.1.1.	Multiplicative Inversion in GF28	64
6.1.2.	Variations on Logical Functions for SubBytes.....	68
6.1.3.	Wolkerstorfer	70
6.1.4.	Boyar.....	72
6.1.5.	Canright	74
6.1.6.	Nogami	76
6.1.7.	Nekado	79
7.	DESIGN FLOW	85
7.1.	DESIGN OF CUSTOM GATES	85
7.1.1.	Schematic.....	87
7.1.2.	Layout	90
7.2.	DIGITAL DESIGN	100
7.2.1.	Algorithmic and Behavioral Description.....	101
7.2.2.	Logic Synthesis.....	101
7.2.3.	Place and Route	103
7.2.4.	Post-Layout Simulation	107
8.	CONCLUSION	135

REFERENCES 138



LIST OF FIGURES

Figure 1.1. DES	4
Figure 1.2. AES	6
Figure 2.1. Distribution of S-box's current consumption at 2 ns	25
Figure 4.1. A differential AND gate	36
Figure 4.2. Differential Cascode Voltage Switch Logic for an AND gate.....	37
Figure 4.3. A dynamic AND gate.....	39
Figure 4.4. A dynamic differential AND gate	40
Figure 4.5. Sense-Amplifier-Based Logic for an AND gate	42
Figure 4.6. SABL with fully connected differential pull-down network for an AND gate.	43
Figure 4.7. SABL with enhanced fully connected differential pull-down network for an AND gate.....	44
Figure 4.8. SABL with enhanced fully connected differential pull-down network for an XNOR gate	45
Figure 4.9. An AND gate implemented with ADDT.....	46
Figure 4.10. An XNOR gate implemented with ADDT	47
Figure 7.1. SABL NAND non-inverting output for PDN widths 420nm-1660nm	88
Figure 7.2. SABL NAND input capacitance for PDN widths 420nm-1660nm	88

Figure 7.3. SABL NAND non-inverting output for PDN widths 420nm-820nm	89
Figure 7.4. SABL NAND non-inverting output for sense-amp widths 420nm-820nm	89
Figure 7.5. SABL NAND n1 node for sense-amp widths 420nm-820nm.....	90
Figure 7.6. SABL NAND	97
Figure 7.7. SABL XNOR	97
Figure 7.8. ADDT NAND	98
Figure 7.9. ADDT XNOR	98
Figure 7.10. Current consumption of <i>Wolkerstorfer_1</i> for 10000 runs	114
Figure 7.11. Correlation coefficients of <i>Wolkerstorfer_1</i> for zero-value model	115
Figure 7.12. Maximum correlation coefficients of <i>Wolkerstorfer_1</i> for zero-value model	116
Figure 7.13. Comparison of key guesses between different implementations	119
Figure 7.14. Comparison of correlation coefficients between different implementations	119
Figure 7.15. Comparison of key guesses between different designs for <i>Canright</i>	120
Figure 7.16. Comparison of correlation coefficients between different designs for <i>Canright</i>	120
Figure 7.17. Comparison of key guesses between different designs for <i>Nekado</i>	121
Figure 7.18. Comparison of correlation coefficients between different designs for <i>Nekado</i>	121

Figure 7.19. Current consumption of <i>Boyar_1</i> for 100 runs	123
Figure 7.20. Key probabilities of <i>Boyar_1</i> via 10000-measurement-strong templates	124
Figure 7.21. Comparison of key guesses between different implementations	125
Figure 7.22. Comparison of correct key probabilities between different implementations	126
Figure 7.23. Comparison of key guesses between different implementations	126
Figure 7.24. Comparison of correct key probabilities between different implementations	127
Figure 7.25. Comparison of key guesses between different implementations	127
Figure 7.26. Comparison of correct key probabilities between different implementations	128
Figure 7.27. Comparison of key guesses between different designs of <i>Nekado</i>	129
Figure 7.28. Comparison of correct key probabilities between different designs of <i>Nekado</i>	129
Figure 7.29. Comparison of key guesses between different designs of <i>Nekado</i>	130
Figure 7.30. Comparison of correct key probabilities between different designs of <i>Nekado</i>	130
Figure 7.31. Comparison of key guesses between different designs of <i>Nekado</i>	131
Figure 7.32. Comparison of correct key probabilities between different designs of <i>Nekado</i>	131
Figure 7.33. Comparison of key guesses between different designs of <i>Canright</i>	132

Figure 7.34. Comparison of correct key probabilities between different designs of *Canright* 132

Figure 7.35. Comparison of key guesses between different designs of *Canright* 133

Figure 7.36. Comparison of correct key probabilities between different designs of *Canright* 133

Figure 7.37. Comparison of key guesses between different designs of *Canright* 134

Figure 7.38. Comparison of correct key probabilities between different designs of *Canright* 134

Figure 8.1. The distinctness of power consumption for zero input 137

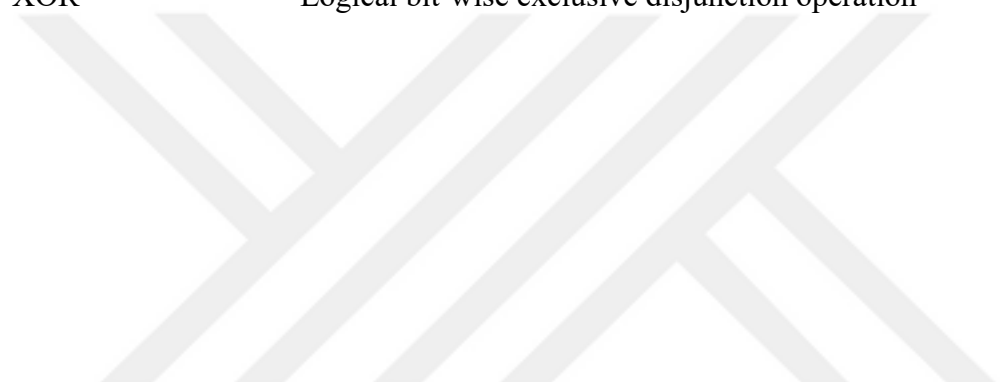
LIST OF TABLES

Table 4.1. Internal nodes of a dynamic differential AND gate during different phases.....	41
Table 7.1. Timing, voltage and load characteristics of D_CELLS.....	85
Table 7.2. Transistor sizes of D_CELLS.....	86
Table 7.3. SABL NAND Sizes.....	93
Table 7.4. SABL XNOR Sizes.....	94
Table 7.5. ADDT NAND Sizes.....	95
Table 7.6. ADDT XNOR Sizes.....	96
Table 7.7. Timing, voltage and load characteristics of SABL and ADDT.....	99
Table 7.8. Circuits created with each design and implementation.....	101
Table 7.9. Gate statistics for the first, third and fourth designs.....	103
Table 7.10. Correlation coefficients and number of measurements estimate for <i>Canright_1</i>	110
Table 7.11. Correlation coefficients and number of measurements estimate for zero-value	112
Table 7.12. Comparison of different hypothetical power consumption models.....	117

LIST OF SYMBOLS/ABBREVIATIONS

ADDT	Another Dynamic Differential Topology
AES	Advanced Encryption Standard
AND	Logical bit-wise conjunction operation
BDD	Binary decision diagram
CASCADE	Chip Architecture for Smart CARds and portable intelligent Devices
CML	Current Mode Logic
CMOS	Complementary metal-oxide-semiconductor
DCVSL	Differential Cascode Voltage Switch Logic
DES	Data Encryption Standard
DPA	Differential power analysis
DSS	Digital Signature Standard
DyCML	Dynamic Current Mode Logic
FPGA	Field-programmable gate arrays
FIPS	Federal Information Processing Standard
GF	Galois field
GND	Ground
IC	Integrated circuit
IPA	Inferential power analysis
MB	Mixed Bases
NAND	Logical bit-wise alternative denial operation
NBS	National Bureau of Standards
NIST	National Institute of Standards and Technology
NOR	Logical bit-wise joint denial operation
NOT	Logical bit-wise negation operation
OR	Logical bit-wise disjunction operation
P-box	Permutation box
PRR	Polynomial Ring Representation
RNG	Random number generator
RRB	Redundantly Represented Basis
RSA	Rivest-Shamir-Adleman cryptosystem

RTZ	Return-to-zero
S-box	Substitution box
SABL	Sense-Amplifier-Based Logic
SNR	Signal-to-noise ratio
SP-network	Substitution-permutation network
SPA	Simple power analysis
VDD	Supply voltage
WDDL	Wave Dynamic Differential Logic
XNOR	Logical bit-wise biconditional operation
XOR	Logical bit-wise exclusive disjunction operation



1. INTRODUCTION

Cryptography (cryptology) in broad terms, is the field of secret writing [1]. The name, cryptography, literally means secret writing in Greek [2]. It studies the techniques of hiding messages from anyone other than their primary targets. These techniques are named ciphers [3] and the study that deals with their analysis and breaching is called cryptanalysis. The secret message is called the plaintext and the encrypted or enciphered message is called the ciphertext. Similarly, obtaining the plaintext from the ciphertext is called deciphering or decrypting. The key can be broadly defined as the secret method that is previously agreed upon which describes a specific way to implement the cipher. This method is chosen among one of the possible methods that the cipher can be implemented with. In theory, ciphertexts encrypted by a key can only be decrypted to meaningful plaintexts by someone else with the knowledge of the same key.

1.1. BRIEF HISTORY OF MODERN CRYPTOGRAPHY

The classical ciphers that are used throughout the history before modern cryptography are generally divided into three major groups [3]. These are concealment ciphers, transposition ciphers and substitution ciphers. The concealment ciphers hide the secret message through any means and mostly rely on the cipher itself not being discovered. They include ciphers such as the puncture cipher and the grille cipher. They just consist of papers with intentionally punched holes on them, which are then aligned with another paper with seemingly random letters written on it, only to reveal the actual secret message when read through the holes [4]. These ciphers are the most primitive ones and do not bear any importance in modern cryptography. The ones that have some contemporary correspondence are the transposition and especially the substitution ciphers. The transposition ciphers do not change the letters of the plaintext but rearrange their positions with a well-defined rule which acts as a key [5][6]. The substitution ciphers operate by substituting each letter of the plaintext by a different letter, numbers or symbols according to a method determined by a key [6][7]. The Caesar cipher is among the most famous examples of simple substitution [8]. It basically shifts each letter of the plaintext up or down a fixed number of letters. This fixed number becomes the key and shifting the letters of the ciphertext in the opposite direction

by as much as this key decipheres the ciphertext. In contrast, the Hill cipher is a polygraphic substitution that is applied by modularly multiplying the plaintext with a matrix that can be considered as the encryption key [9]. The decryption is accomplished in this case by modularly multiplying the ciphertext by the inverse of the matrix used in encryption which then becomes the decryption key. What is interesting about the Hill cipher is the fact that it is similar to the `MixColumns` operation of the Advanced Encryption Standard (AES) algorithm in that they both utilize matrix multiplications. With these being told, the extent of classical ciphers' relevance is very limited. For the most part, they are already extensively cryptanalyzed and easily deciphered without keys.

Serious mathematical approaches to cryptography began after Claude Elwood Shannon's paper [10] that explores cryptography from an information theoretical point of view. In his paper, he reiterated Kerckhoffs' principle that a cryptosystem's security should not depend on the secrecy of its implementation, but only the secrecy of its key [11][12]. Furthermore, based on his analysis of the ciphers that existed at the time, he mentioned two properties, diffusion and confusion, that would determine a cryptosystem's strength. He defined diffusion as the dissipation of plaintext's redundancies throughout the ciphertext. This makes it so that the patterns of the language and repetitions are averaged over the ciphertext. Meanwhile, confusion is defined in his paper as diminishing of the statistical relationship between the key and the ciphertext. As a result, confusion serves to ensure that posterior probabilities of keys do not depend on the resulting ciphertext. These and the others definitions in Shannon's paper laid the foundation of modern cryptography [13].

Later, the 1970s saw the invention of asymmetric-key algorithms by Whitfield Diffie and Martin E. Hellman [14]. Every cryptographic algorithm up to that point has been a symmetric-key algorithm in which there is a single key for both encryption and decryption. In contrast, asymmetric-key algorithms utilize two distinct keys, one for each of encryption and decryption. They have two main areas of application: one is called public-key cryptosystems and the other is called the one-way authentication systems.

In public-key cryptosystems, a party has a publicly available encryption key that any party can use to encrypt, but a secret decryption key that only he or she can use to decrypt. Ideally, the decryption key cannot be calculated from the encryption key. The motivation behind this stems from a need to securely share keys that belong to symmetric-key algorithms. A

theoretical key exchange protocol that fulfills this task is presented in Diffie and Hellman's paper [14], later named Diffie-Hellman key exchange protocol [15]. In general, security of symmetric-key algorithms can be more well-defined. Therefore, a typical secure communication scenario between two parties would involve a key exchange through a public key algorithm followed by standard exchange of messages through a symmetric-key algorithm.

As for one-way authentication systems, their purpose is to verify the source of a message. A sender encrypts the message using his or her secret encryption key, and this message can be decrypted by anyone in possession of the public decryption key. Instead of hiding the plaintext from third parties, these systems broadcast the message to everyone and act as signature or authentication tools.

The other cryptologic development in 1970s happened when National Bureau of Standards (NBS) declared the symmetric-key algorithm Data Encryption Standard (DES) as a standard encryption system of the United States government [16]. Following this standardization by a United States institution, DES was cryptanalyzed thoroughly by the international academic community. Thanks to this situation, it was able to be established as a secure algorithm for its time despite controversies [17].

DES has a repetitive and symmetric structure [18]. Its repetitions are called rounds and its symmetry exists between its encryption and decryption algorithms. It is a block cipher, meaning it is applied to blocks of n -bit plaintexts as opposed to being applied bit by bit to a stream of plaintexts. It enciphers a 64-bit plaintext block using a 56-bit key. It has 16 rounds, each of which uses a unique round key derived from the primary 56-bit key. Its general structure for encryption is visualized in Figure 1.1.

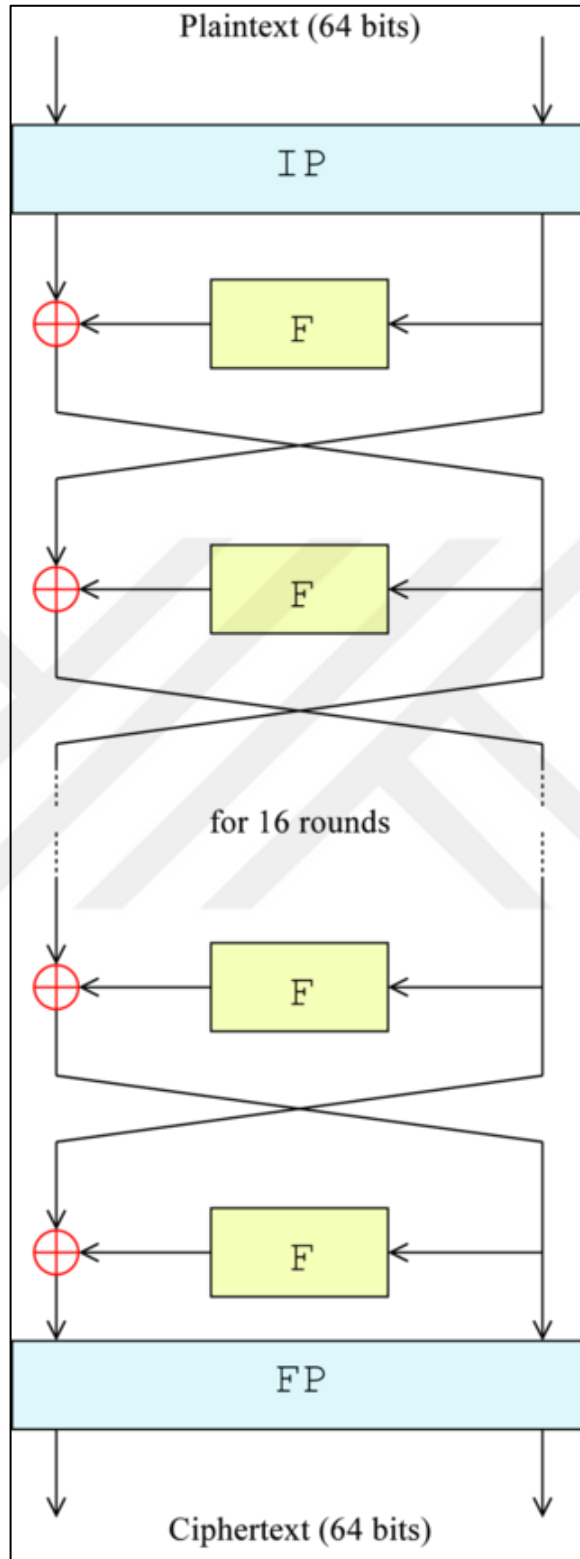


Figure 1.1. DES

The \oplus operator in Figure 1.1 corresponds to an XOR operation, the IP and FP blocks correspond to the initial and final permutations, and the F block corresponds to the Feistel or round function that uses the round keys. The initial and final permutations bear no cryptographic importance, but only act as an interface for the 1970s hardware [16]. The Feistel function is the most critical block of DES. It contains a substitution-box (S-box) and a permutation-box (P-box) connected in series. The S-box is a non-linear transformation which is implemented in DES by mapping the input to an output according to a look-up table. The P-box is a linear transformation that transposes bits of the input. The S-box and P-box equip the algorithm respectively with confusion and diffusion properties mentioned in Shannon's paper [10]. The overall structure of Figure 1.1 that consists of splitting the plaintext, applying the round function to one half, XORing the result with the other half and swapping the halves for the next round is called the Feistel network [19]. The main advantage of the Feistel structure is that decryption can be accomplished using the same encryption algorithm given in Figure 1.1 by way of only reversing the order of the round keys and swapping the initial permutation with the final permutation. Hence, decryption does not require the inverse of the round function, which results in ease of implementation.

With the millennium approaching, technology was catching up and new cryptanalytical methods were being discovered. On that account, National Institute of Standards and Technology (NIST), formerly known as NBS, began a selection process for a new symmetric-key encryption standard in 1997 [20][21]. This process ended in 2001 with the selection of Rijndael algorithm as the Advanced Encryption Standard (AES) [22]. A schematic that roughly represents the encryption of AES is given in Figure 1.2 [23].

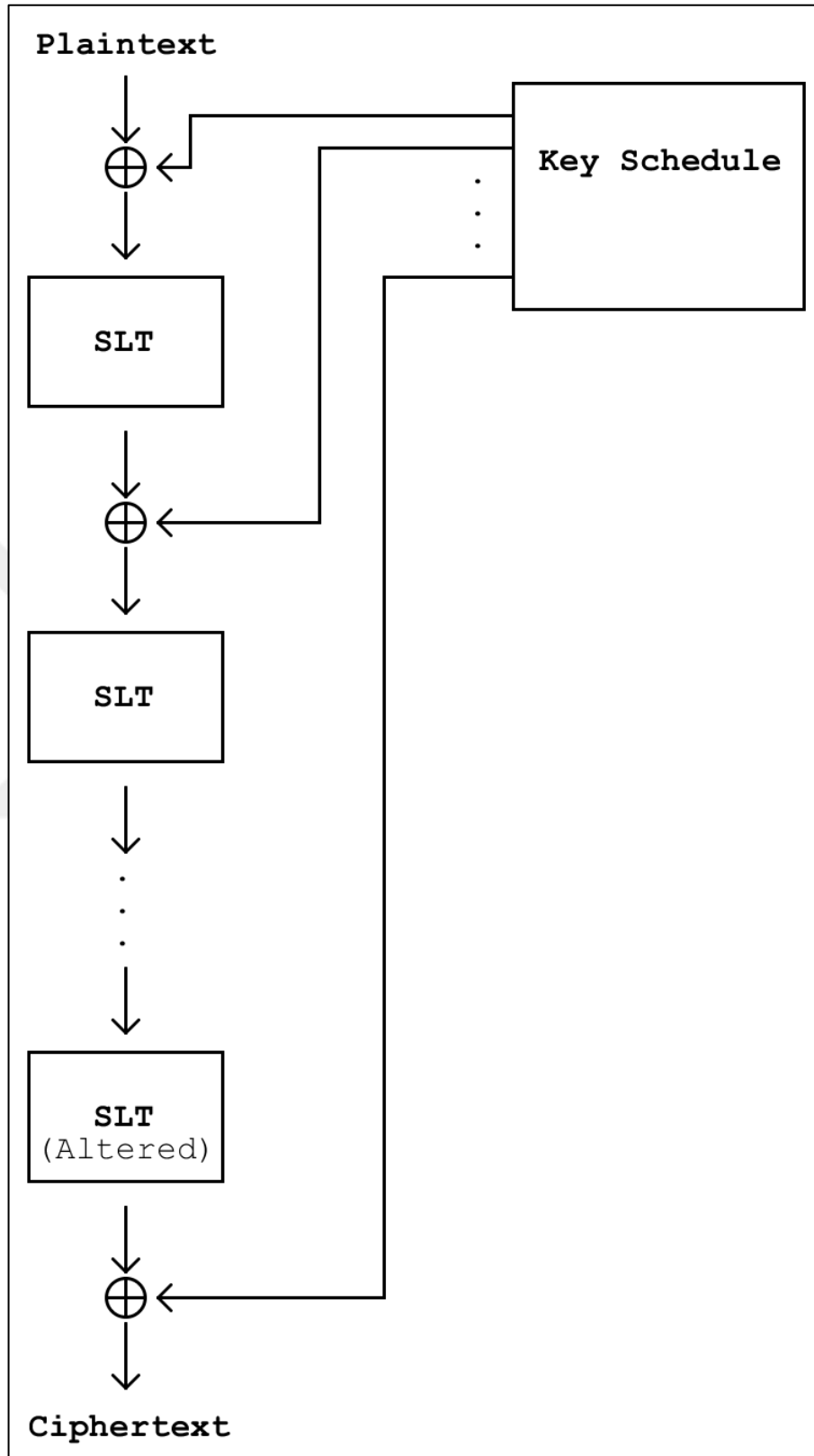


Figure 1.2. AES

The \oplus operator in Figure 1.2 corresponds to an XOR operation and the SLT block corresponds to the round function. The round function is an S-box followed by a P-box and an additional linear transformation. Similar to DES, the function of S-box is to provide the algorithm with confusion, while the subsequent linear transformations accomplish the same with diffusion. The structure of Figure 1.2 that includes successive application of round function and XORing of the round key is called a substitution-permutation network (SP-network) [24]. SP-networks differ from Feistel networks in that they require the inverse of the SLT block for decryption. Thus, they require separate operations for decryption. But their advantage over Feistel networks is that they apply their round function to the plaintext as a whole in each round instead of after multiple rounds, and since the round functions are the only parts of each algorithm that support parallelism, SP-networks benefit more from parallel computing [25].

The following sections will focus on the S-box of AES with relevance to this thesis. Certain terms and concepts will be mentioned without being delved into in the interest of explaining them in detail further into the text. Most of the discussion will be kept about the work in literature on the subject.

1.2. AES S-BOX

Contrary to DES, S-box of AES is not just a look-up table in essence, although it can be designed as such in practice. The S-box of AES is a finite field arithmetic operation followed by an affine transformation [26]. This nature of AES S-box allows a diversity of implementations.

The finite field arithmetic operation in AES S-box is a multiplicative inversion in $GF(2^8)$. However, calculating the multiplicative inverse in $GF(2^8)$ without a look-up table is not efficient. Therefore, it is implemented in what is called composite (towering) fields [27][28]. Different implementations also diverge in their finite field representations. Finite fields are similar to vector spaces, in that they can be represented as a linear combination of a set of vectors called basis. There are two common finite field bases called the polynomial basis and the normal basis. The multiplicative inverse can be calculated in any finite field as long as it is isomorphic to AES $GF(2^8)$. This way it can be mapped from and to AES $GF(2^8)$

before and after the calculation of the multiplicative inverse. Having said that, some of the S-box implementations in literature is listed below together with the method they employ to calculate the multiplicative inverse.

- A. Rudra et al. [28], S. K. Mathew et al. [29] and J. Wolkerstorfer et al. [30] calculate the inverse in $GF((2^4)^2)$ and use polynomial basis to represent both $GF(2^4)$ and $GF((2^4)^2)$.
- Y. S. Jeon et al. [31] base their design on the one presented by J. Wolkerstorfer et al. [30]. Their contribution is revamping the design to a resourced shared version for both encryption and decryption.
- A. Hodjat et al. [32] calculate the inverse in $GF((2^4)^2)$ and use polynomial basis to represent both $GF(2^4)$ and $GF((2^4)^2)$. They further try to increase the throughput by pipelining the S-box via inserting registers.
- X. Zhang et al. [33] calculate the inverse in $GF((2^4)^2)$ and use polynomial basis to represent both $GF(2^4)$ and $GF((2^4)^2)$. Instead of the inversion in $GF(2^4)$ they try to simplify the multiplication in $GF(2^4)$ by representing it as $GF((2^2)^2)$. Finally, they look into the options of increasing the throughput by pipelining the S-box via inserting registers.
- R.R. Rach et al. [34] base their study on the same S-box design given in X. Zhang et al. [33]. They then apply their own reductions upon that design.
- S. Nikova et al. [35] calculate the inverse in $GF((2^4)^2)$ and use normal basis to represent both $GF(2^4)$ and $GF((2^4)^2)$.
- X. Zhang et al. [36], A. Satoh et al. [37][38] and N. Mentens et al. [39] calculate the inverse in $GF(((2^2)^2)^2)$ and use polynomial basis to represent $GF(2^2)$, $GF((2^2)^2)$ and $GF(((2^2)^2)^2)$.
- S. Kumar et al. [40] and M. M. Wong et al. [41] calculate the inverse in $GF(((2^2)^2)^2)$ and use polynomial basis to represent $GF(2^2)$, $GF((2^2)^2)$ and $GF(((2^2)^2)^2)$. Furthermore, they apply certain reductions that significantly modify the operations in $GF(2^2)$ and $GF((2^2)^2)$.
- M. Mozaffari-Kermani et al. [42] calculate the inverse in $GF(((2^2)^2)^2)$ and use normal basis to represent $GF(2^2)$, $GF((2^2)^2)$ and $GF(((2^2)^2)^2)$. Furthermore, they apply certain reductions that significantly modify the operations in $GF(2^2)$ and $GF((2^2)^2)$.

- D. Canright [43][44] calculates the inverse in $GF(((2^2)^2)^2)$ and uses normal basis to represent $GF(2^2)$, $GF((2^2)^2)$ and $GF(((2^2)^2)^2)$. Additionally, he conducts a comprehensive research of all the possible bases for $GF(((2^2)^2)^2)$ and each of its subfields.
- Y. Nogami et al. [45] calculate the inverse in $GF(((2^2)^2)^2)$. They use a method called Mixed Bases (MB) that provides flexibility in the choice of basis for $GF(2^2)$, $GF((2^2)^2)$ and $GF(((2^2)^2)^2)$.
- K. Nekado et al. [46] calculate the inverse in $GF((2^4)^2)$. In addition to using MB method of Y. Nogami et al. [45], they introduce a new representation called Redundantly Represented Basis (RRB) for $GF(2^4)$.
- R. Ueno et al. [47] calculate the inverse in $GF((2^4)^2)$ and use normal basis to represent $GF((2^4)^2)$. Apart from this, they employ RRB representation of K. Nekado et al. [46] to calculate the multiplication in $GF(2^4)$ and their own representation dubbed Polynomial Ring Representation (PRR) to calculate the multiplicative inverse in $GF(2^4)$.
- M. Mozaffari-Kermani et al. [48] make an extensive comparison of S-box implementations in literature up to that point.
- L. Batina et al. [49] attempt to optimize the number of flip-flops in terms of throughput when pipelining the AES S-box. To this end, they make use of the genetic algorithm method.
- A. Dogan et al. [50] put forward introduction of registers between the inner operations of AES S-box with the purpose of reducing the dynamic power consumption.
- R. Ueno et al. [51] propose an architecture that unifies encryption and decryption of the S-box in an efficient manner.
- S. Morioka et al. [52] use an architecture named twisted-BDD (Binary Decision Diagram) that is based on logical reduction of the loop-up table.
- K. Rahimunnisa et al. [53] apply logical reduction techniques such as Karnaugh map to the standard look-up table.
- J. Boyar et al. [54][55] calculate the inverse in $GF(((2^2)^2)^2)$ and use normal basis to represent $GF(2^2)$, $GF((2^2)^2)$ and possibly $GF(((2^2)^2)^2)$. They then apply a logic minimization technique that splits the inversion's linear and non-linear

components. Finally, they attempt to reduce the number of AND gates of the non-linear part and the number of XOR gates of the linear parts.

- Z. Wang et al. [56] suggest that the cryptographic security of AES would not diminish if AES $GF(2^8)$ was abandoned entirely in favor of calculating the multiplicative inverse natively in a composite field. Nevertheless, the resulting algorithm is not AES.

Comparison of these implementations suggest a general trend that calculating the multiplicative inverse in $GF(((2^2)^2)^2)$ results in more compact circuits than calculating it in $GF((2^4)^2)$. The implementation of D. Canright [43] is a very efficient one in that regard. However, D. Canright [43] is limited in the choice of basis conversion matrices compared to Y. Nogami et al. [45], because their MB method allows a multiplication or inversion in $GF((2^m)^n)$ to have its input represented in different basis than its output. It is important to have conversion matrices with as few ones as possible, since they are multiplied with the input and output of the multiplicative inversion operator. As exceptions, even though K. Nekado et al. [46] and R. Ueno et al. [47] find the multiplicative inverse in $GF((2^4)^2)$, they use special representations for $GF(2^4)$, therefore they can be said to be more efficient than the implementations of D. Canright [43] and Y. Nogami et al. [45]. Additionally, some logic minimization techniques such as the one used by J. Boyar et al. [54] are among the most efficient implementations. Besides all that, many of the implementations make their own trade-offs between critical path delay and total number of gates. Eventually, realization of any implementation heavily depends on the process technology and the logic synthesis tools.

1.3. SIDE-CHANNEL ATTACKS

Although AES is strong algorithmically, real world implementations have other variables such as time and energy to consider. These variables make the system vulnerable from alternative angles. One such angle is a non-invasive method called side-channel attacks. The most popular side-channel attacks are power analysis, electromagnetic and timing attacks [57]. These attacks exploit the statistical relationship of a cryptographic circuit's operation with its power consumption, electromagnetic emanations and execution time in order. This thesis focuses on the most popular one among these, power analysis attacks. Most of the research is found upon the book on the subject by S. Mangard et al. [58]. With that in mind,

a list of papers concerning power analysis and its countermeasures is given below as reference.

- The first appearance of power analysis attacks in literature was in the paper written by P. Kocher et al. [59] in 1999. It defines simple power analysis (SPA) as a visual inspection of power measurements. Meanwhile, the paper applies differential power analysis (DPA) in two steps. In the first step, it targets one of the eight S-boxes in the 16th round of DES, then computes one bit of this input for each key hypothesis. In the second step, it calculates the difference of mean power measurements for when the input bit is one when the input bit is zero for separately each key hypothesis. The paper claims for incorrect key hypotheses, the power measurements will be grouped randomly and the difference will be very small.
- P.C. Kocher [60] cryptanalyzes symmetric-key algorithms such as Diffie-Hellman, RSA and DSS by showing that encryption time of systems employing these algorithms depend on the key.
- J.F. Dhem et al. [61] applies timing attack on a smart card implementation of RSA called CASCADE.
- P.N. Fahn et al. [62] put forward a new power analysis technique called Inferential Power Analysis (IPA). Their technique contains a profiling stage in which the power consumption of a cryptographic device is measured while it performs 100s of encryptions. Afterwards the statistical characteristics of these measurements are characterized. In the next stage, the secret key of a new device of the same type is extracted from a single power trace by comparing it to the profile that has been created in the previous stage.
- S. B. Örs et al. [63] experiment with success of differential power analysis on elliptic-curve cryptosystems implemented on FPGAs. Elliptic-curve cryptography is a public-key cryptography variant that is based upon groups formed by the points on an elliptic curve defined over a finite field and the lines that intersect them [64].
- K. Schramm et al. [65] utilize power analysis to carry out a collision attack on the intermediate values of DES. Collision attacks are a method that relies on finding a ciphertext for which two distinct plaintexts are encrypted to with the same key [66]. This happens when the cipher is not bijective and reduces the possible set of secret keys.

- D. Agrawal et al. [67] study electromagnetic information leakage of various symmetric-key and asymmetric-key implementations.
- F.X. Standaert et al. [68] try to quantify the information leakage of a device and its exploitability by attackers in a widely-applicable way. They then test the effectiveness of countermeasures such as masking and noise addition on Hamming weight leakage with their newly created metric. This unified benchmark is then expanded by F.X. Standaert et al. [69].
- S. Chari et al. [70] introduce a very strong approach called template attacks. Their method is similar to P.N. Fahn et al. [62], it consists of a characterization and a comparison phase. In the characterization phase, power consumption of a device is measured while it performs many encryptions for different known values of an internal operation. The value of this internal operation should be associated with the key. Afterwards, a probability density function, named a template, for each of these known values is calculated by assuming Gaussian distribution. During the comparison phase, a small number of power measurements are taken from another device of the same kind with an unknown key. Finally, the probability of these measurements belonging to a template is calculated for every template according to maximum likelihood estimation.
- W. Schindler et al. [71] present what is called the stochastic method. Analogous to the method in S. Chari et al. [70] they characterize the power consumption of a device prior to mounting an attack. They treat the power consumption as being comprised of two parts, namely a data dependent deterministic part and a random noisy part. To represent the noisy part, they use a probability density function of a random distribution. However, unlike S. Chari et al. [70], they characterize the deterministic part as a function of an internal operation's value. This function is a linear equation with the internal operation's bits as variables. During the characterization phase, this linear equation's coefficients are calculated by solving a system of equations in which the coefficients are variables, the bits are coefficients and the equations are equal to the power measurements. During an attack on another identical device, maximum likelihood estimation is carried out using this device's power measurements and the previously calculated equation and probability density function of noise.

- B. Gierlichs et al. [72] compare template attacks and stochastic method. They then try to make their own improvements on both techniques. In summary, they make a crude conclusion that template attacks are more successful if there are a lot of power measurements available during the characterization phase, although in situations when the number of characterization measurements is low, stochastic method has a better accuracy.
- D. Agrawal et al. [73] describe an attack that exploits a combination of multiple side-channel leakages, namely power and electromagnetic.
- K. Tiri et al. [74] draw attention to the fact that models used to simulate the power consumption of a device for a designer is very influential in precisely determining its power analysis resistance before tape-out. To do so, they compare power consumption simulations of different capacitance extraction models against power measurements from a real device in the context of power analysis.
- K.J. Kulikowski et al. [75] show that many gate level power analysis countermeasures that balance the power consumption of different transitions, look over the propagation delay discrepancy between different transitions. In their paper they prove this vulnerability by simulating an attack against a topology that take preventative actions against differing propagation delays and another that does not.
- R. Elbaz et al. [76] examine the cryptosystems where an external memory, as in separate from the IC where the processing is done, exists and the communication data bus between the processing unit and the memory unit can be compromised. They mention workarounds that encrypt the data transferred on these external buses.
- A. Yu et al. [77], Z.C. Yu et al. [78], S. Guilley [79], F. Gürkaynak et al. [80] and S. Moore et al. [81] advocate the adoption of self-timed (asynchronous) circuits in tandem with other countermeasures to thwart both power analysis attacks via misalignment and timing attacks.
- K.J. Kulikowski et al. [82] highlight the fact that Return-To-Zero (RTZ) protocols reduce the possible set of input transitions, and therefore in implementations where power consumption of registers is not the dominant component, make the device less resistant to power analysis attacks by diminishing the randomness of the combinational parts of the circuit where it is applied. They point out that this even eliminates the advantages gained by employing a dual-rail or an asynchronous topology.

- Z. Toprak et al. [83] argue that the dynamic power consumption of synchronous CMOS logic at clock edges is among the main contributors of their vulnerability to power analysis attacks. As a solution they propose using Current Mode Logic (CML) [84] because of its more static power consumption and add that for high frequencies, its average power consumption is comparable to classic CMOS logic.
- F. Mace et al. [85] state that Dynamic Current Mode Logic (DyCML) [86] exhibit the same power analysis resistance characteristics of CML. In addition, contrary to CML, DyCML gates work in a precharge-evaluation cycle that overcomes the constant current consumption of CML.
- K. Tiri et al. [87] present a dynamic differential topology called Sense-Amplifier-Based Logic for power analysis resistance which is based on Sense-Amplifier-Based Flip-Flop [88] and similar to Differential Cascode Voltage Switch Logic (DCVSL) [89]. Gates built with this topology have two phases: precharge and evaluation. As a result of both having RTZ property and being differential, for every input combination, exactly one transition happens at the outputs of SABL.
- K. Tiri et al. [90] progress the SABL topology by proposing a modification to its pull-down network by the name of Enhanced Fully-Connected Differential Pull-Down Network (DPDN) which addresses internal net charge imbalances, uneven ground path resistances and switching delay disparity problems of regular SABL DPDNs.
- K. Tiri et al. [91] suggest a way to realize differential and RTZ logic for the purpose of power analysis resistance with logic gates found in every standard cell library. They accomplish this by using gates that realize monotonic Boolean functions that have complementary outputs given complementary inputs. They call this topology Wave Dynamic Differential Logic (WDDL). When the input of a WDDL circuit is set to all zeros, it propagates through every WDDL gate like a wave to reset all of them. Afterwards, a valid input causes a transition in only one output of every WDDL gate in the same manner. This mode of operation stems from WDDL's monotonicity. When an input of a monotonic Boolean function transitions from zero to one, its output cannot transition from one to zero, and equivalently when one of its inputs makes a transition from one to zero, its output cannot transition from zero to one.
- D. Sokolov et al. [92] explore the effects of having two spacers, such as all zeros and all ones instead of one spacer in contrast to RTZ, on power analysis resistance. They

test this method on different actualizations of dual-rail logic assembled from ordinary logic gates.

- M. Aigner et al. [93] propose a new single-rail dynamic logic style dubbed 3state Dynamic Logic (3sDL) with three different levels and charge sharing scheme with a dummy capacitor equal to the output capacitance. The third level acts as a reset value during the precharge phase and is ideally right in the middle of VDD and GND. Owing to this third level and charge sharing of the output capacitance with the dummy capacitor, two capacitances are alternately charged to and discharge from half of VDD during evaluation and precharge phases. This assures equal power consumption for every input transition without needing differential logic.
- T. Sundström et al. [94] does an extensive comparison between various logic gate topologies including static CMOS, SABL, DCVSL and DyCML. They conclude in their analysis that DyCML gives the best security while DCVSL is very effective despite its relatively low complexity.
- P. Corsonello et al. [95] suggest that power should be supplied to the cryptographic circuit using a charge pump in order to isolate the external power consumption from the internal power consumption and accordingly make the circuit more power-analysis-resistant.
- G.B. Ratanpal et al. [96], D. Mesquita et al. [97] and P. Rakers et al. [98] advocate connecting an analog current regulator with feedback to the power supply pins of cryptographic circuits in order to desensitize overall power consumption from power fluctuations caused by internal operations. This is roughly accomplished by creating a current sink for when the current consumption of the circuit is low, so it is kept relatively constant.
- N. Pramstaller et al. [99] and C. Herbst et al. [100] offer a power analysis countermeasure called masking that differs from others by mostly being hinged on mathematics. In masking, a random value called mask that is most preferably independent and uniformly distributed is combined with the sensitive values calculated during encryption or decryption. This way, the power consumption of the device becomes random and thus independent from these sensitive values. They describe this concept algorithmically in their papers.
- T. Popp et al. [101] and J.Dj. Golic et al. [102] show topologies that apply masking on logic gate level.

- K. Schramm et al. [103] present a means to counter higher order power analysis attacks by higher order masking. Higher order power analysis aims to eliminate the effects of masking by combining the power measurements of two different intermediate values masked with the same mask. The paper offers using even more masks that are independent from each other to circumvent the attack.

Power analysis attacks are very popular in the scientific community, because they can totally undermine the security of strong algorithms by attacking their weak points, they are non-invasive and really easy to execute. The SPAs can only unveil the device's rudimentary operation order. The DPAs that try to find the statistical relationships between hypothetical measurements and real measurements are able to exploit even the smallest data dependencies, yet are fairly limited by the attackers' ability to guess the working principles of the device. The strongest attacks are the DPAs that profile the device exhaustively beforehand, such as the template-based attacks. They do not lean on hypothetical models and work according to very simple rules of probability theory.

When it comes to countermeasures, there exist many in the literature with their own disadvantages. Architectural level countermeasures such as asynchronous designs are highly complex and deviate too much from the most commonplace practices. Cell level masking countermeasures are complex in their own right and depend on the randomness of the Random Number Generator (RNG) hardware that generates the masks. Cell level hiding techniques get easier to implement but less power-analysis-resistant the more they resemble static CMOS logic. CML/DyCML is not that similar to static CMOS, have high power consumption (static in regards to CML and dynamic in regards to DyCML) and require the use of additional resistors and/or capacitors. Dynamic differential logic takes up at least twice the area, consumes twice the power and requires clocking the combinational logic parts of the circuit. The standard library solutions also suffer from the same consequences that Dynamic differential logic suffers except the clock related ones, and they are also the least power-analysis-resistant ones. The isolation solutions are truly implementation-independent, but they either utilize large capacitors that take up area or analog circuitry that increases power consumption.

1.4. OBJECTIVE OF THE THESIS

This thesis initially intends to examine all aspects of various AES S-box implementations that are based on finite field arithmetic rather than Boolean reduction or look-up tables. Afterwards, these implementations are used as a basis for power analysis resistant designs applied at the logic cell level, namely dynamic differential logic. The end goal is to observe the impact of S-box implementation and the logic gate style on power analysis resistance at once. To do this, numerous variations of a subcircuit of AES that only contains the S-box part is built and simulated in an Electronic Design Automation (EDA) program. The simulated power consumptions are then processed to resemble power consumption of a real device. These simulated and processed measurements are then subjected to correlation-based and template-based attacks. As a consequence, it is seen that all the finite field arithmetic implementations are equally resistant to power analysis. Therefore, the implementation can be chosen according to other criteria. The use of dynamic differential on the other hand renders correlation-coefficient based attacks useless and makes the circuit ten to a hundred times more resistant to template-based attacks.

All that being said, the text is structured as follows:

- Second chapter gives a background on sample statistics and how to estimate certain parameters and discriminate parameters of two different populations.
- Third chapter describes two power analysis attack techniques, correlation-based and template-based attack, using the definitions in the previous chapter.
- Fourth chapter briefly tells about existing countermeasures and presents two dynamic differential topologies.
- Fifth chapter explains the concept of finite fields starting from very primitive definitions and arriving at the definitions of normal basis and polynomial basis.
- Sixth chapter outlines the AES algorithm and specifies the S-box part in detail with the help of the definitions in the preceding chapter.
- Seventh chapter gives information about the whole design process, the simulation methodology, the attack procedure and the results.
- The last chapter summarizes and comments on the findings and gives impressions about the outcomes.

2. STATISTICAL BACKGROUND

The whole foundation of power analysis is statistics. The power measurements of a device are the samples of the population that comprises every possible value of power it can consume. The measurements are a way to depict the probability density of this population. This section defines well-known statistical quantities and parameters that would aid in describing the power measurements of a device and guessing how well these measurements correlate with the operations inside the device. The definitions below can be found in many textbooks, but they mainly follow the definitions of the book written by S. Mangard et al. [58]

2.1. NORMAL DISTRIBUTION

A random variable with normal (Gaussian) distribution is defined by two parameters: mean (μ) and variance (σ^2). Probability distribution function (PDF) of normal distribution is given in (2.1) in terms of μ and σ^2 and random variable x .

$$PDF = f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (2.1)$$

Since it is not possible to know the mean and variance of an infinite population, it is more relevant to talk about sample mean ($\hat{\mu}$) and unbiased sample variance (s^2) defined as in (2.2) and (2.3).

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.2)$$

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \hat{\mu})^2 \quad (2.3)$$

x_i in (2.2) and (2.3) is the i th sample of the normally distributed random variable X .

Unbiased sample covariance between two sample sets of normally distributed random variables X and Y is formulized in (2.4).

$$q_{X,Y} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \hat{\mu}_X)(y_i - \hat{\mu}_Y) \quad (2.4)$$

Pearson correlation coefficient between the aforementioned sample sets is given in (2.5).

$$r_{X,Y} = \rho(X,Y) = \frac{q_{X,Y}}{\sqrt{s_X^2 s_Y^2}} = \frac{\sum_{i=1}^n (x_i - \hat{\mu}_X)(y_i - \hat{\mu}_Y)}{\sqrt{\sum_{i=1}^n (x_i - \hat{\mu}_X)^2} \sqrt{\sum_{i=1}^n (y_i - \hat{\mu}_Y)^2}} \quad (2.5)$$

2.2. ESTIMATION OF PARAMETERS

Sample parameters are all estimates of population parameters. Thus, they are random variables with specific distributions. $\hat{\mu}$ of n variables is normally distributed with the following mean (expected value) and variance in (2.6) and (2.7).

$$E(\hat{\mu}) = \mu \quad (2.6)$$

$$Var(\hat{\mu}) = \frac{\sigma^2}{n} \quad (2.7)$$

s^2 of n variables is a chi-squared distribution with expected value and variance given in (2.8) and (2.9).

$$E(s^2) = \sigma^2 \quad (2.8)$$

$$Var(s^2) = \frac{2 \times \sigma^4}{n-1} \quad (2.9)$$

$r_{X,Y}$ on the other hand, has a complicated distribution by itself. Its Fisher Transformation $\tanh^{-1}(r_{X,Y})$ though has approximately a normal distribution with expected value and

variance shown in (2.10) and (2.11). Obviously, the expected value of sample correlation coefficient r is equal to population correlation coefficient ρ as demonstrated in (2.10).

$$E(\tanh^{-1}(r_{X,Y})) = \tanh^{-1}(\rho_{X,Y}) = \frac{1}{2} \ln \frac{1 + \rho_{X,Y}}{1 - \rho_{X,Y}} \quad (2.10)$$

$$\text{Var}(\tanh^{-1}(r_{X,Y})) = \frac{1}{n-3} \quad (2.11)$$

2.3. CONFIDENCE INTERVALS

Confidence intervals are estimates that a particular parameter is within specific limits with a certain probability. As an example, consider the nearly normal distribution of $\tanh^{-1}(r_{X,Y})$. If one were to subtract its mean from itself and divide it by its standard deviation, one would get a standard normal distribution with $\mu = 0$ and $\sigma^2 = 1$. So W in (2.12) is a standard normal distribution.

$$W = \left(\frac{1}{2} \ln \frac{1 + r_{X,Y}}{1 - r_{X,Y}} - \frac{1}{2} \ln \frac{1 + \rho_{X,Y}}{1 - \rho_{X,Y}} \right) \sqrt{n-3} \quad (2.12)$$

Now, since the probability of a standard normal variable being inside an interval bounded by $[-z, z]$ is as given in (2.13), with $z = \Phi^{-1}(1 - \alpha/2)$, where Φ is the cumulative distribution function and α is the error probability, writing the expression in (2.12) in place of x in (2.13) results in the equation (2.14) and (2.15).

$$p(-z \leq x \leq z) = 1 - \alpha = \Phi(z) - \Phi(-z) \quad (2.13)$$

$$p\left(-z \leq \left(\frac{1}{2} \ln \frac{1 + r_{X,Y}}{1 - r_{X,Y}} - \frac{1}{2} \ln \frac{1 + \rho_{X,Y}}{1 - \rho_{X,Y}} \right) \sqrt{n-3} \leq z\right) = 1 - \alpha \quad (2.14)$$

$$\begin{aligned} p\left(\frac{1}{2} \ln \frac{1 + r_{X,Y}}{1 - r_{X,Y}} - \frac{z}{\sqrt{n-3}} \leq \frac{1}{2} \ln \frac{1 + \rho_{X,Y}}{1 - \rho_{X,Y}} \leq \frac{1}{2} \ln \frac{1 + r_{X,Y}}{1 - r_{X,Y}} + \frac{z}{\sqrt{n-3}}\right) \\ = 1 - \alpha \end{aligned} \quad (2.15)$$

This means $\mu = \tanh^{-1}(\rho_{X,Y})$ of $\tanh^{-1}(r_{X,Y})$ is inside the interval given by (2.16) with a probability of $1 - \alpha$.

$$\left[\frac{1}{2} \ln \frac{1 + r_{X,Y}}{1 - r_{X,Y}} - \frac{z}{\sqrt{n-3}}, \frac{1}{2} \ln \frac{1 + r_{X,Y}}{1 - r_{X,Y}} + \frac{z}{\sqrt{n-3}} \right] \quad (2.16)$$

2.4. NUMBER OF SAMPLES

The following expressions describe the number of samples required to estimate the parameters ρ and $\rho_0 - \rho_1$ up to a given accuracy.

2.4.1. Number of Samples for ρ

Assume that probability of absolute distance between sample correlation coefficient and population correlation coefficient being more than a constant is as given in (2.17).

$$p \left(\left| \frac{1}{2} \ln \frac{1 + r_{X,Y}}{1 - r_{X,Y}} - \frac{1}{2} \ln \frac{1 + \rho_{X,Y}}{1 - \rho_{X,Y}} \right| > \frac{1}{2} \ln \frac{1 + c}{1 - c} \right) = \alpha \quad (2.17)$$

Then the following equations in (2.18), (2.19), (2.20) and (2.21) can be derived.

$$p \left(\left| \frac{1}{2} \ln \frac{1 + r_{X,Y}}{1 - r_{X,Y}} - \frac{1}{2} \ln \frac{1 + \rho_{X,Y}}{1 - \rho_{X,Y}} \right| \sqrt{n-3} > \frac{1}{2} \ln \left(\frac{1 + c}{1 - c} \right) \sqrt{n-3} \right) = \alpha \quad (2.18)$$

$$p \left(|W| > \frac{1}{2} \ln \left(\frac{1 + c}{1 - c} \right) \sqrt{n-3} \right) = \alpha \quad (2.19)$$

$$2p \left(W > \frac{1}{2} \ln \left(\frac{1 + c}{1 - c} \right) \sqrt{n-3} \right) = \alpha \quad (2.20)$$

$$p \left(W < \frac{1}{2} \ln \left(\frac{1 + c}{1 - c} \right) \sqrt{n-3} \right) = \Phi \left(\frac{1}{2} \ln \left(\frac{1 + c}{1 - c} \right) \sqrt{n-3} \right) = 1 - \alpha/2 \quad (2.21)$$

Rewriting (2.21) for n provides the expression in (2.22).

$$n = 3 + 4 \frac{(\Phi^{-1}(1 - \alpha/2))^2}{\ln^2 \frac{1+c}{1-c}} \quad (2.22)$$

Which is the number of samples n necessary for the difference between sample correlation coefficient and population correlation coefficient to be less than or equal to c .

Another important definition is the number of samples necessary to distinguish $r_{X,Y}$ from zero. It can be deduced by either presuming $\tanh^{-1}(\rho_{X,Y}) < 0$ and computing $p(\tanh^{-1}(r_{X,Y}) < 0)$ or doing the same for $\tanh^{-1}(\rho_{X,Y}) > 0$ and $p(\tanh^{-1}(r_{X,Y}) > 0)$. Deriving it for the former as shown in (2.23), (2.24), (2.25), (2.26) and (2.27) yields the expression in (2.28).

$$p\left(\frac{1}{2} \ln \frac{1+r_{X,Y}}{1-r_{X,Y}} < 0\right) = 1 - \alpha \quad (2.23)$$

$$p\left(\frac{1}{2} \ln \frac{1+r_{X,Y}}{1-r_{X,Y}} - \frac{1}{2} \ln \frac{1+\rho_{X,Y}}{1-\rho_{X,Y}} < -\frac{1}{2} \ln \frac{1+\rho_{X,Y}}{1-\rho_{X,Y}}\right) = 1 - \alpha \quad (2.24)$$

$$\begin{aligned} p\left(\left(\frac{1}{2} \ln \frac{1+r_{X,Y}}{1-r_{X,Y}} - \frac{1}{2} \ln \frac{1+\rho_{X,Y}}{1-\rho_{X,Y}}\right) \sqrt{n-3} \right. \\ \left. < -\left(\frac{1}{2} \ln \frac{1+\rho_{X,Y}}{1-\rho_{X,Y}}\right) \sqrt{n-3}\right) = 1 - \alpha \end{aligned} \quad (2.25)$$

$$\Phi\left(-\left(\frac{1}{2} \ln \frac{1+\rho_{X,Y}}{1-\rho_{X,Y}}\right) \sqrt{n-3}\right) = 1 - \alpha \quad (2.26)$$

$$\Phi^{-1}(1 - \alpha) = -\left(\frac{1}{2} \ln \frac{1+\rho_{X,Y}}{1-\rho_{X,Y}}\right) \sqrt{n-3} \quad (2.27)$$

$$n = 3 + 4 \frac{(\Phi^{-1}(1 - \alpha))^2}{\ln^2 \frac{1+\rho_{X,Y}}{1-\rho_{X,Y}}} \quad (2.28)$$

2.4.2. Number of Samples for $\rho_0 - \rho_1$

$\tanh^{-1}(r_0) - \tanh^{-1}(r_1)$ (calculated with equal number of samples n) has a normal distribution with expected value and variance given in (2.29) and (2.30).

$$E\left(\frac{1}{2}\ln\frac{1+r_0}{1-r_0} - \frac{1}{2}\ln\frac{1+r_1}{1-r_1}\right) = \frac{1}{2}\ln\frac{1+\rho_0}{1-\rho_0} - \frac{1}{2}\ln\frac{1+\rho_1}{1-\rho_1} \quad (2.29)$$

$$\text{Var}\left(\frac{1}{2}\ln\frac{1+r_0}{1-r_0} - \frac{1}{2}\ln\frac{1+r_1}{1-r_1}\right) = \frac{2}{n-3} \quad (2.30)$$

This brings one to an important definition in terms of power analysis. How many samples are required to distinguish $\tanh^{-1}(r_0)$ from $\tanh^{-1}(r_1)$? By applying the same procedure to the assumption $\tanh^{-1}(\rho_0) - \tanh^{-1}(\rho_1) < 0$ and finding $p(\tanh^{-1}(r_0) - \tanh^{-1}(r_1))$ one gets the expression in (2.35) through the deduction process demonstrated in (2.31), (2.32), (2.33) and (2.34).

$$p\left(\frac{1}{2}\ln\frac{1+r_0}{1-r_0} - \frac{1}{2}\ln\frac{1+r_1}{1-r_1} < 0\right) = 1 - \alpha \quad (2.31)$$

$$p\left(\left(\frac{1}{2}\ln\frac{1+r_0}{1-r_0} - \frac{1}{2}\ln\frac{1+r_1}{1-r_1}\right) - \left(\frac{1}{2}\ln\frac{1+\rho_0}{1-\rho_0} - \frac{1}{2}\ln\frac{1+\rho_1}{1-\rho_1}\right) < -\left(\frac{1}{2}\ln\frac{1+\rho_0}{1-\rho_0} - \frac{1}{2}\ln\frac{1+\rho_1}{1-\rho_1}\right)\right) = 1 - \alpha \quad (2.32)$$

$$p\left(\left[\left(\frac{1}{2}\ln\frac{1+r_0}{1-r_0} - \frac{1}{2}\ln\frac{1+r_1}{1-r_1}\right) - \left(\frac{1}{2}\ln\frac{1+\rho_0}{1-\rho_0} - \frac{1}{2}\ln\frac{1+\rho_1}{1-\rho_1}\right)\right] \sqrt{\frac{n-3}{2}} < -\left(\frac{1}{2}\ln\frac{1+\rho_0}{1-\rho_0} - \frac{1}{2}\ln\frac{1+\rho_1}{1-\rho_1}\right) \sqrt{\frac{n-3}{2}}\right) = 1 - \alpha \quad (2.33)$$

$$\Phi^{-1}(1 - \alpha) = -\left(\frac{1}{2}\ln\frac{1+\rho_0}{1-\rho_0} - \frac{1}{2}\ln\frac{1+\rho_1}{1-\rho_1}\right) \sqrt{\frac{n-3}{2}} \quad (2.34)$$

$$n = 3 + 8 \frac{(\Phi^{-1}(1 - \alpha))^2}{\left(\ln \frac{1 + \rho_0}{1 - \rho_0} - \ln \frac{1 + \rho_1}{1 - \rho_1}\right)^2} \quad (2.35)$$

Φ and Φ^{-1} correspond to cumulative distribution function and its inverse for a standard normal distribution, while α is the probability of error in every preceding equation.

2.5. STATISTICAL PROPERTIES OF POWER CONSUMPTION

Instantaneous power consumption of a circuit can be categorized into separate and independent components from the power analysis point of view. There are basically three relevant components of power at any specific instant. These are the exploitable part (P_{exp}), operating noise (P_{opn}) and electronic noise (P_{eln}). Exploitable part is defined as any part of the power consumption that depends on the targeted operation and data of the attack. Operating noise is the power consumption resulting from the remaining irrelevant parts of the circuit that run in parallel at that moment. Electronic noise component includes noise from every other source, namely parasitic circuit components, faultiness of the measurement setup, heat, electromagnetic radiation etc.

One could assume all these components to be independently and normally distributed. This assumption is easily accepted in the case of the parts that constitute noise. The exploitable part may also be approximated by a normal distribution as it is demonstrated in Figure 2.1. Distribution of S-box's current consumption at 2 ns, where it represents the current consumption of an S-box circuit at a certain instant. While separating power consumption into different components, mean of the total power consumption, which amounts to the sum of these independent distributions' means, may be described as a separate constant term (P_{const}). But since it is constant, it is ineffectual with regard to variance-based analyses, hence could be left out of discussion in those cases. This leaves three independent distributions with zero means that represent the variance of power consumption at an instance between different runs.

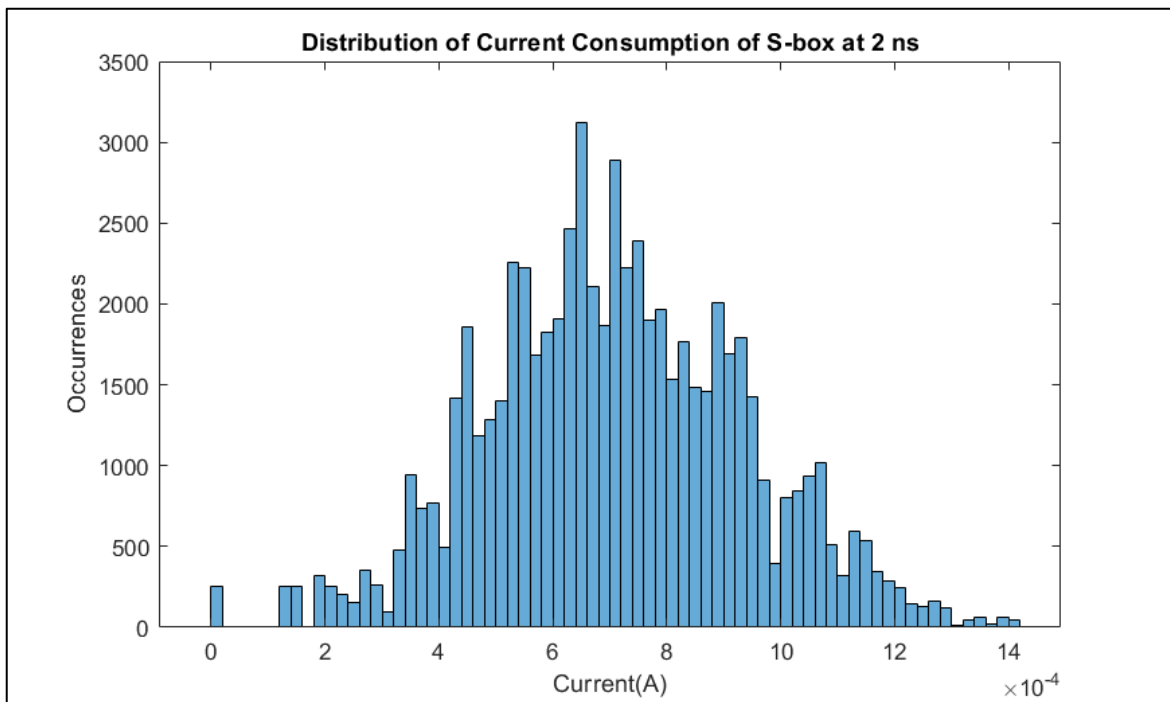


Figure 2.1. Distribution of S-box's current consumption at 2 ns

While making this distinction of different components, another important point is that they are not strictly defined in terms of what they are made of. As an example; exploitable part can include operating noise and operating noise can include electronic noise in it, but not vice versa. So, the grouping is started with incorporating every part that correlates to the attacked data and operation and naming it the exploitable part. The rest can be collectively categorized as noise or further differentiated into independent parts.

One universal factor that determines how much information is leaked through power consumption is signal-to-noise ratio in which signal is designated as the exploitable information. The ratio then becomes the ratio of exploitable parts' variance to noise variance as given by equation (2.36).

$$SNR = \frac{Var(P_{exp})}{Var(P_{opn} + P_{eln})} \quad (2.36)$$

3. POWER ANALYSIS

Power analysis is a decryption method for cryptographic circuits that takes advantage of the fact that power consumption leaks information about the operation of digital circuits. As a result, it is classified as a side-channel attack, meaning that it is not a direct attack on the cryptographic algorithm itself. The information digital circuits leak is mainly because of the switching that occurs inside them depending on the operations and data they operate on. It is obvious that different inputs and different logical functions lead to different logical levels at input, output and internal nets. Additionally, real logic gates with propagation delays greater than zero introduce glitches to the system. These glitches attribute further characteristics to each possible input transition.

Power analysis attacks, since their discovery, have come to be known by two types in literature. These are named simple power analysis and differential power analysis. The distinction between them is in the number of power measurements utilized during the attack. It can be inferred from the name that simple power analysis uses one or very few power measurements and sometimes relies on visual inspection. Usually, this merely gives some clues about the general structure of the cryptographic circuit and is ineffective with the exception of very primitive AES implementations on software. Differential power analysis on the other hand utilizes two measurements at minimum and relies on statistics. Appropriately, power analysis attacks of this project all fall into the differential power analysis category.

3.1. CORRELATION COEFFICIENT-BASED ATTACK

The principle behind correlation coefficient-based attack is creating hypothetical power consumptions that depend on a certain intermediate value of the attacked device and calculating Pearson correlation coefficients between these and actual power measurements from the device. The attack can be classified as a known plaintext or known ciphertext attack, meaning that the sole information attacker has is either the plaintext being enciphered or the ciphertext after encryption. As an example, here is how a known plaintext attack is carried out:

- The power consumption of the device-under-attack is recorded as it enciphers p plaintexts with an unknown key. While doing this, the plaintexts should be as uniformly distributed as possible so that the exploitable part of the power consumption has larger variance.
- The plaintexts are put in a $1 \times p$ matrix P , while the measurements are sampled in time and inserted into a matrix M of size $p \times t$, where the power consumption of each plaintext enciphered are in the rows of the matrix M and placed in the same order as in the matrix P (power measurements of $plaintext_j$ in column j of P being enciphered is in j th row of M), while the columns represent samples in time, needless to say, ordered chronologically.
- An intermediate value that is a function of plaintext and key is chosen according to certain requirements that will be mentioned later. In the case of AES, the best intermediate value for a known plaintext attack is the output of the S-box in the first round specified by $Sub(plaintext \oplus key)$.
- For each possible key dubbed key hypothesis, this intermediate value is calculated adopting every plaintext in matrix P as inputs. This calculation leads to a matrix called V . The size of the matrix is $k \times p$ where the row i and column j contains the intermediate value $f(key_i, plaintext_j)$.
- A number is assigned to each intermediate value according to a certain power consumption model. This assignment transforms V into hypothetical power consumption matrix H , where each element corresponds to the power consumption attributed to that intermediate value.
- Lastly, Pearson correlation coefficients between each row of H and each column of M is calculated according to equation (2.5) to form a matrix C of size $k \times t$. The $(i, j)^{th}$ element of this matrix represents the correlation coefficient between key hypothesis k_i at time t_j .

Visual representations of matrices described above are supplied below in (3.1), (3.2), (3.3), (3.4) and (3.5).

$$P = [plaintext_1 \quad \cdots \quad plaintext_p] \quad (3.1)$$

$$M = \begin{bmatrix} power_{1,1} & \cdots & power_{1,t} \\ \vdots & \ddots & \vdots \\ power_{1,p} & \cdots & power_{p,t} \end{bmatrix} \quad (3.2)$$

$$V = \begin{bmatrix} f(key_1, plaintext_1) & \cdots & f(key_1, plaintext_p) \\ \vdots & \ddots & \vdots \\ f(key_k, plaintext_1) & \cdots & f(key_k, plaintext_p) \end{bmatrix} \quad (3.3)$$

$$H = \begin{bmatrix} power_{1,1}^{hypothetical} & \cdots & power_{1,p}^{hypothetical} \\ \vdots & \ddots & \vdots \\ power_{k,1}^{hypothetical} & \cdots & power_{k,p}^{hypothetical} \end{bmatrix} \quad (3.4)$$

$$C = \rho(H_i, M_j) = \begin{bmatrix} correlation_{1,1} & \cdots & correlation_{1,t} \\ \vdots & \ddots & \vdots \\ correlation_{k,1} & \cdots & correlation_{k,t} \end{bmatrix} \quad (3.5)$$

$i = 1, 2, \dots, k$
 for and
 $j = 1, 2, \dots, t$

The C matrix constructed through this procedure contains correlation information between every possible key and the power consumption of device-under-attack at every discrete time instance. So depending on how accurate the model for hypothetical power consumption values is and how many and how uniform the measurements are made, one would expect the elements of C to be equally small at everywhere except at (i, j) , where the row suggests correct key hypothesis and the column is near the moment power consumption has the strongest dependency to the intermediate value.

3.1.1. Hypothetical Power Consumption Models

Besides signal-to-noise ratio, one of the main designators of correlation coefficient-based attacks' success is how well the hypothetical power consumption values represent the real ones. Without the knowledge of a cryptographic device's netlist, the attacker can just create power consumption models based on the AES algorithm or simple guesses.

For instance, if the attacked device writes the intermediate value on a register with high input or output capacitance, any write operation momentarily contributes to a noticeable increase in power consumption. This creates a vulnerability point, importance of which depends on

how much power is consumed during such a write operation. This kind of vulnerability is easily exploited with a correlation coefficient-based attack using primitive power consumption models. These include models based on transitions in any bit or all bits of the intermediate value.

AES S-box is random enough that coming up with complicated hypothetical models is not possible without the detailed knowledge of the netlist. So, in a typical case, finding the most suitable model comes down to trial and error. One could start by testing if the device leaks Hamming weight of the intermediate value. If that does not work, the next step could be to try whether any bit correlates with the power consumption. Finally, since most `SubBytes` implementations either have low or at least distinct power consumption characteristics for the input value of zero, one can create a model that separates that input value from others and see if that correlates with the power measurements from the device. From now on, these models are going to be called Hamming weight model, bit models and zero-value model.

3.2. TEMPLATE-BASED ATTACK

Template-based attacks are a kind of side-channel attack that are established around the idea of representing the power consumption of a device by an ideal random distribution for each state of the intermediate value, and then attempting to guess the correct key based on which state the successive power measurements are more likely to be associated with.

These representations belonging to different states, are called templates. They are created by using a sample device, which should have the same circuit as the attacked device, so that it has the same power consumption characteristics. By way of making numerous measurements at specific points in time while the sample device encrypts known plaintexts with known keys, a data population is created, and approximated by a random distribution to form the templates. Afterwards, measurements from the attacked device are compared against each template and assigned to the most likely one. Since each template represents a key or a set of keys, fitting the measurements to one of the templates equates to finding the key or reducing the possible set of correct keys. Similar to correlation coefficient-based attacks, template attacks can be known plaintext or known ciphertext attacks depending on the intermediate value that is exploited.

It was mentioned before that the power consumption of a cryptographic circuit at a certain instant between separate runs is well-approximated by a standard normal distribution. Therefore, the templates can be created by fitting normal distribution curves to the data acquired for each intermediate value state. What is meant by fitting here is actually the calculation of mean and covariance matrices (\mathcal{M} and Σ) given in (3.6) and (3.7) where X_i is the distribution of i^{th} point of the subset of points taken from each measurement, $\hat{\mu}$ is the sample mean in equation (2.2), s^2 is the sample variance in equation (2.3) and q is the covariance in equation (2.4).

$$\mathcal{M} = [\hat{\mu}_{X_1} \quad \hat{\mu}_{X_2} \quad \cdots \quad \hat{\mu}_{X_n}] \quad (3.6)$$

$$\Sigma = \begin{bmatrix} q_{X_1, X_1} = s_{X_1}^2 & \cdots & q_{X_1, X_n} \\ \vdots & \ddots & \vdots \\ q_{X_n, X_1} & \cdots & q_{X_n, X_n} = s_{X_n}^2 \end{bmatrix} \quad (3.7)$$

The intermediate value states are basically the intermediate value itself or functions of it. To give an example, for the output of the S-box in the first round as the intermediate value, the templates can be based on its Hamming weight, one of its bits or itself as a whole. In the case of picking its Hamming weight, one has to create nine distinct templates for nine values its Hamming weight take. In order to represent one of the output bits, one needs two create two templates. And finally, if the templates are based on the output value itself, there needs to be 256 distinct templates. It is obvious that, this decision determines the degree to which the subset of possible correct keys can be reduced assuming the attacks is successful. To point it out, successfully attributing the measurements from the attacked device to one of the two templates that depend on a bit results in eliminating half of the keys as incorrect. Meanwhile, a successful pairing with one of the 256 templates associated with the hexadecimal value of the output directly reveals the correct key. Yet, it should also be acknowledged that creating 256 templates takes way more effort and time than creating only two templates. Lastly, while deciding what the templates are going to be based on, it should be remembered that the templates only need to be created once, whereas they can be used however many times as necessary.

In the attacking phase, the power measurements of the attacked device are ranked from most likely to least likely in terms of which template's probability distribution function they might be sampled from. This ranking is organized in agreement with Bayes' rule. In terms of

Bayes' rule, the thing being ranked here is the probability of a key or set of keys encompassing the correct key, given a set of power measurements acquired from the device under attack. This posterior probability can be denoted by $p(\text{template}_i|M)$, in which template_i is the i^{th} template and M is the set of measurements from the attacked device. Writing Bayes' rule for $p(\text{template}_i|M)$ gives the expression in (3.8).

$$\begin{aligned}
p(\text{template}_i|M) &= \frac{p(M|\text{template}_i)p(\text{template}_i)}{p(M)} \\
&= \frac{p(M|\text{template}_i)p(\text{template}_i)}{\sum_x p(M|\text{template}_x)p(\text{template}_x)} \\
&= \frac{p(\text{power}_1, \text{power}_2, \dots, \text{power}_j, \dots, \text{power}_p|\text{template}_i)p(\text{template}_i)}{\sum_x p(\text{power}_1, \text{power}_2, \dots, \text{power}_j, \dots, \text{power}_p|\text{template}_x)} \quad (3.8) \\
&= \frac{\prod_j p(\text{power}_j|\text{template}_i) p(\text{template}_i)}{\sum_x (\prod_j p(\text{power}_j|\text{template}_x))p(\text{template}_x)} \\
&= \frac{\prod_j p(\text{power}_j|\text{template}_i)}{\sum_x (\prod_j p(\text{power}_j|\text{template}_x))}
\end{aligned}$$

Rewriting $p(M|\text{template}_i)$ in (3.8) as $\prod_j p(\text{power}_j|\text{template}_i)$ is appropriate only when power_j , j^{th} measurement in M is independent from others for every j . Therefore, it is really crucial that the measurements are taken from the attacked device as it encrypts plaintexts as randomly as possible. Finally, the simplification in the last step of (3.8) can be made because the prior probability of every key is equal.

If the distribution of measurements is approximated by a normal distribution, $p(\text{power}_j|\text{template}_i)$ in (3.8) can be calculated via the multivariate variant of the PDF in equation (2.1). This variant is given below in (3.9) where \mathfrak{X} is a row vector of (x_1, x_2, \dots, x_n) that correspond to the points picked out of j^{th} power measurement power_j as displayed in (3.10) and the pair \mathcal{M} and Σ are the mean and covariance matrices of the i^{th} template template_i .

$$\begin{aligned}
PDF_{\text{multivariate}} &= f(x_1, x_2, \dots, x_n|\mathcal{M}, \Sigma) \\
&= \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(\mathfrak{X}-\mathcal{M})\Sigma^{-1}(\mathfrak{X}-\mathcal{M})^T} \quad (3.9)
\end{aligned}$$

$$\mathbf{x} = [x_1 \quad x_2 \quad \cdots \quad x_n] \quad (3.10)$$

This method of finding $template_i$ that maximizes $p(template_i|M)$ is called the maximum likelihood estimation.

3.2.1. Choosing the Points of Interest

To build templates as indicated in the last section, one has to choose points in each measurement data so that sample distributions are accumulated that can then be approximated by ideal random distribution functions. There are a lot of choices in this step and not a single solution exists. There are a lot of data points to pick from, in addition to the freedom of processing this data as seen fit. One could pick a single point from each power measurement, or pick multiple points to resemble a multivariate random distribution. Moreover, points can be added, integrated, averaged and so on. All of this is done so as to reduce the number of data points required to build the templates. Ideally, it would yield better results to use as many data points as possible to create the templates. But the same can be said about the number of measurements as well. As it stands, picking more data points from more measurements extends the template building, and consequently the key extraction duration. Ergo, a point of equilibrium ought to be found that includes variables such as time, accuracy, number of measurements, number of data points and the way these points are combined.

Combining and processing of the data points would only give better results in the existence of a time-axis misalignment or when a higher order differential power analysis is in question. For the reasons that will be apparent in the sections to come, these two cases are not a subject of this text. Apropos of selecting between a single point and multiple points, even though it is unmistakable that more points introduce more overhead, multiple points that are picked at fairly apart positions do a better job of exploiting distributed leakages. Having this in mind, this project utilizes a multivariate random distribution approach based on five variables. The choice of number five here is part arbitrary and part from literature. These five variables are certain points in the power measurements that include the point of maximum variance, the points that are estimated to correlate best with the correct key and some randomly chosen points.

3.3. CHOOSING AN INTERMEDIATE VALUE

First and foremost, an intermediate value has to be easily calculable from plaintext or ciphertext, and key. The function that produces it needs to be nonlinear enough so that even small changes in its input lead to substantial differences in its output. This generates variations in the output regardless of similarities between separate inputs. Therefore, intermediate values do not mix-up with each other as much and correlate better with hypothetical power consumptions.

Two best candidates for AES are the S-boxes in the first and the last round. If the attack is going to be a known or chosen plaintext type, it is very straight forward to pick and apply it to one of the S-boxes in the first round. The first round S-boxes accept 8-bit portions of $plaintext \oplus key$ as their inputs. Hence by choosing its output, one would have an intermediate value that is directly a function of the key and the plaintext. For a known or chosen ciphertext attack, one could use $SubBytes^{-1}(ShiftRows^{-1}(ciphertext \oplus key_{last\ round}))$ as an intermediate value. This would additionally require calculation of the original key from the last round key.

For the reasons stated above and simplicity, this project focuses on the S-boxes of AES alone. To this end, it seeks to create an imaginary scenario in which the output of the first S-box is attacked by correlation coefficient-based and template-based attack methods. For this purpose, a number of different circuits are created, all of which are equivalent when it comes to realizing the S-boxes of AES, and treated as the S-boxes in the first round.

4. DESIGNS AGAINST POWER ANALYSIS

As mentioned before power analysis depends on the correlation between power consumption and logical operations within the circuit. The most compelling solutions to breaking up this correlation would be to create a circuit that consumes constant or random amount of power while performing cryptographic functions.

Another thing power analysis depends on is alignment of power traces recorded, or more generally appropriate matching of the samples that belong to different power traces, while the circuit performs different encryptions. It can be seen from the application of power analysis attacks from the previous section, that one sample of each power trace is treated as if they belong to the same statistical population. This is only accurate if they actually coincide with the different instances of exactly the same encryption operation performed. Otherwise the sample does not correctly represent one single population, reducing the correlation with the correct hypothesis.

4.1. TIME DOMAIN COUNTERMEASURES

The best countermeasure to be taken against power analysis in terms of its natural randomness and lack of unnecessary overhead is building an asynchronous cryptographic circuit. However, it is an entirely separate and substantial subject on its own, contents of which lay outside of the scope of this thesis.

As far as synchronous cryptographic circuits are concerned, one could either change the order of operations or insert dummy operations to misalign different instances of encryption in time domain. But changing the order of operations is limited to the capabilities of the algorithm and inserting additional operations introduces unwanted delays.

All things considered, since this project solely focuses on S-boxes of AES circuits, the discussion is limited to a couple of clock periods. Thus, the countermeasures involving time domain is not further explored from here on.

4.2. AMPLITUDE DOMAIN COUNTERMEASURES

In the amplitude domain, the main goal is to make the power consumption of the circuit independent of the data being operated on by either making it invariant or random for every value of those data.

One solution is to isolate the circuit from the power supply by the help of a low-pass filter that evens out the current fluctuations, a charge pump that feeds the circuit with current internally after being charged by the power supply or a current regulator that consumes current even when the circuit does not. Nonetheless, the low-pass filters and charge pumps use capacitors that take up a lot of area, while the current regulators waste too much power.

The other mentioned approach, that is the randomizing of power consumption, is called masking. Masking is based on secret sharing [104][105], which is a method of sharing a secret by splitting it into parts. Each part, called a share, is useless on its own, but when it is combined with all the other shares, the secret is revealed. In masking, the intermediate value becomes the secret. It can only be revealed by combining the masked intermediate value and the mask. The masked intermediate value is acquired by applying the mask to the intermediate value via an operation such as XOR or multiplication. As long as the mask is uniformly and independently random the masked intermediate value is uncorrelated to the intermediate value. Masking, as defined here mathematically, can be implemented in software. The way to implement it on custom hardware is often achieved through specially designed logic gates that calculate its output depending on the value of the mask. The one apparent difficulty of masking approach is the generation of the random mask. It requires an addition of an RNG circuit and a whole mask distribution network akin to a clock generator and a clock distribution network. Which is why it is out of the scope of this thesis.

The approach of creating a circuit, power consumption of which is data invariant is called hiding. A straightforward way to fulfill this objective on a cell level is to design logic gates that consume the same amount of power regardless of its inputs. Two fundamental logic styles used for this purpose are the CML and dynamic differential logic. The main components of CML are a differential pair and a load. Its logic is provided by the polarity of small differential voltages and the differential pair is connected to a current sink that sinks a mostly constant amount of current. Ergo, compared to static CMOS logic, it consumes so

much less dynamic power. Notwithstanding, its static power consumption is high and even though other low power variants such as DyCML exist they require additional capacitors and are quite complex.

All this discussion brings one to the dynamic differential logic. The philosophy behind it is to have an equal number of input and output transitions for every possible input combination. This is attained by representing each signal with a difference of two signals and sequentially setting and resetting every signal. All this is explained below in detail.

4.2.1. Dynamic Differential

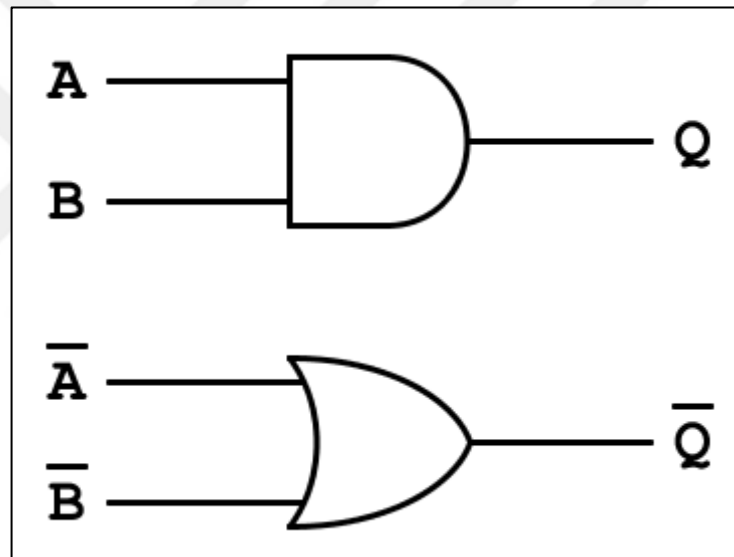


Figure 4.1. A differential AND gate

Differential (dual-rail) logic gates have complementary inputs and outputs. Which means every input and output is treated as a voltage difference between two wires and these two wires should always be logical NOT of each other. This modus operandi can be realized using DeMorgan's Theorems to create two complementary gates. An example of a differential AND gate is given in Figure 4.1 and (4.1). A differential logic gate topology called Differential Cascode Voltage Switch Logic (DCVSL) for the gate in Figure 4.1 is shown in Figure 4.2. It uses two complementary pull-down networks, whose outputs are controlling each other's pull-up transistor in a positive feedback configuration. Since the

pull-up load only consists of a single transistor instead of a network, DCVSL gates have comparable footprint to static CMOS logic gates. Usage of differential logic gates eliminates the need for inverters, as inverting a signal can be achieved by just swapping the complementary wires. However, DCVSL has increased dynamic power consumption compared to static CMOS.

$$\begin{aligned} Q &= A \wedge B \\ \bar{Q} &= \overline{A \wedge B} = \bar{A} \vee \bar{B} \end{aligned} \quad (4.1)$$

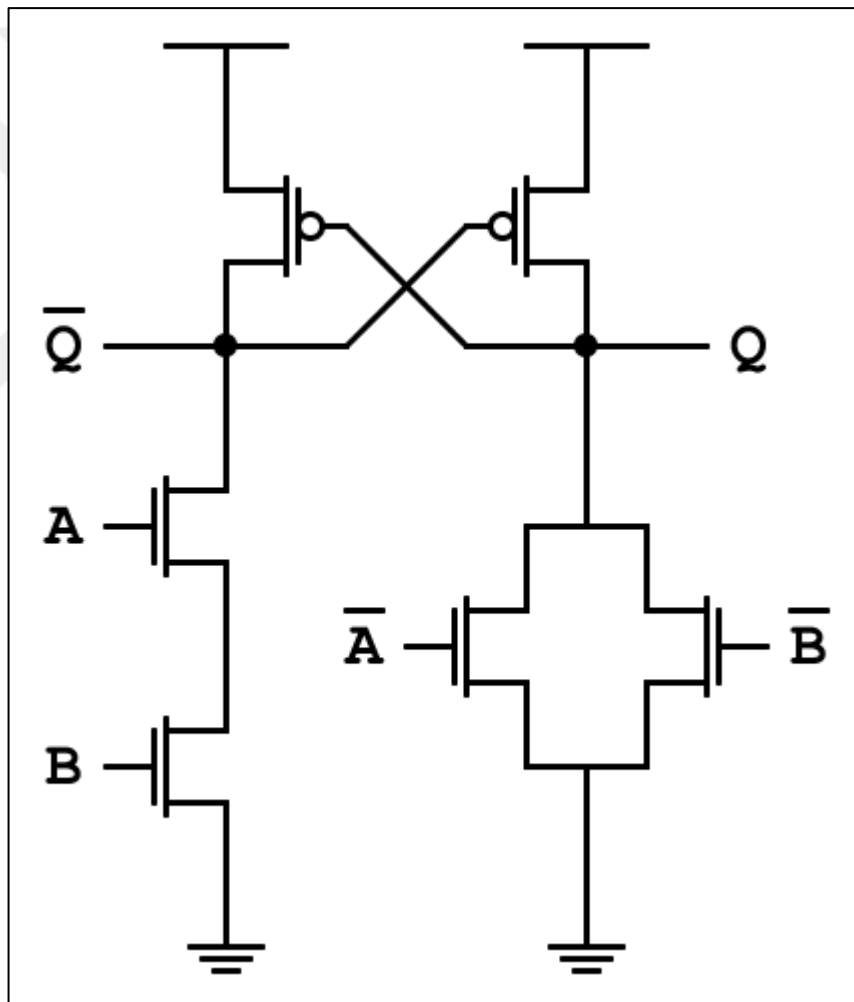


Figure 4.2. Differential Cascode Voltage Switch Logic for an AND gate

Differential logic gates have some benefits in terms of power analysis too. For example, an output transition from one to zero and zero to one both result in transitions in each of the

complementary wires. So, if the complementary wires of the output have equal loads, an equal amount of charge will be stored and emptied when an output transition occurs. However, number of input transitions may vary for different output transitions causing disparities in power consumption. Additionally, amount of total charging and discharging that happens at the internal nodes may still create an unbalance. Finally, and most importantly, input combinations that result in no output transition will very likely consume different amount of energy compared to the ones that do.

To combat these issues, a system should be created in which there is an equal number of input and output transitions for every input and output combination. This can be achieved by periodically introducing a reset value for every complementary wire between different inputs. That way, even if inputs and/or outputs do not change between two instances, there will be a transition from a reset value to a valid value for one of the complementary wires.

Dynamic logic gates can be used to attain this goal. They work in a precharge-evaluation cycle that is controlled by a clock. An example of a dynamic AND gate is given in Figure 4.3. This example exhibits Domino logic, which is the rationale behind the inverter at the output. It is among the various methods used for the purposes of cascading dynamic logic gates. The operation of Figure 4.3 can be explained this way: regardless of **A** and **B**, **Q** is set to logical low at the falling edge of the clock signal, as **M3** is in conduction and **M4** is in cut-off, which signifies the precharge phase. At the rising edge of the clock signal, **Q** becomes $A \wedge B$, thus the evaluation phase begins.

Every time a dynamic logic gate enters the precharge phase, its output is set to a predetermined value. When this is coupled with a dual-rail logic style, at each evaluation phase one of the complementary wires of each input and output will transition from the precharge value, hence half the external nodes will charge or discharge. If the logic gates are designed so that complementary wires of each input are connected to equal number of transistors, equal amount of charge transfer will take place externally for every input combination.

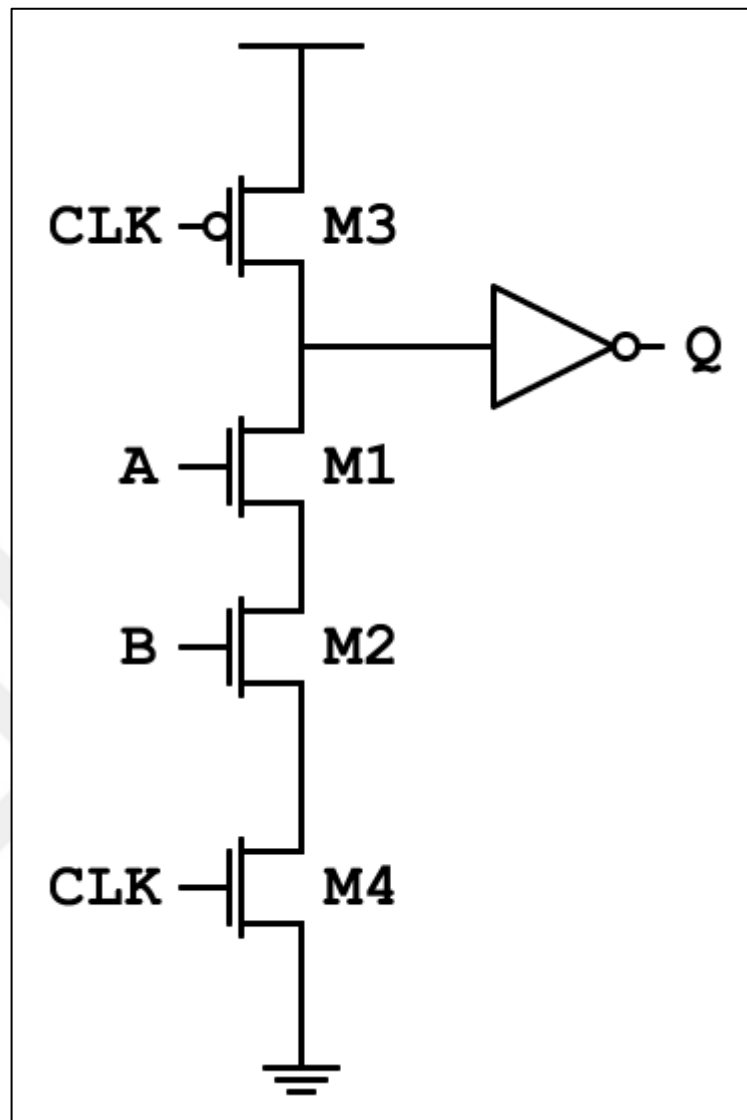


Figure 4.3. A dynamic AND gate

As an example, dynamic logic of Figure 4.3 combined with DCVSL topology of Figure 4.2 is demonstrated in Figure 4.4. When the **CLK** signal is logical low, the circuit enters the precharge phase and both **Q** and its complement become logical low. If it is assumed that **A**, **B** and their complements are outputs of other dynamic differential gates they become logical low as well. Consequently, at the end of a precharge phase, every external port of Figure 4.4 settles to logical low. This resets the previously stored values and prevents glitches. After **CLK** switches to logical high, driven by other gates, one of the complementary wires for both inputs is raised to logical high. This in turn drives one of the complementary outputs high. **M8** and **M9** act as positive feedback charge restoration transistors for when the pull-down

network remains disconnected for a long time and their W/L ratios are smaller compared to **M5** and **M6**.

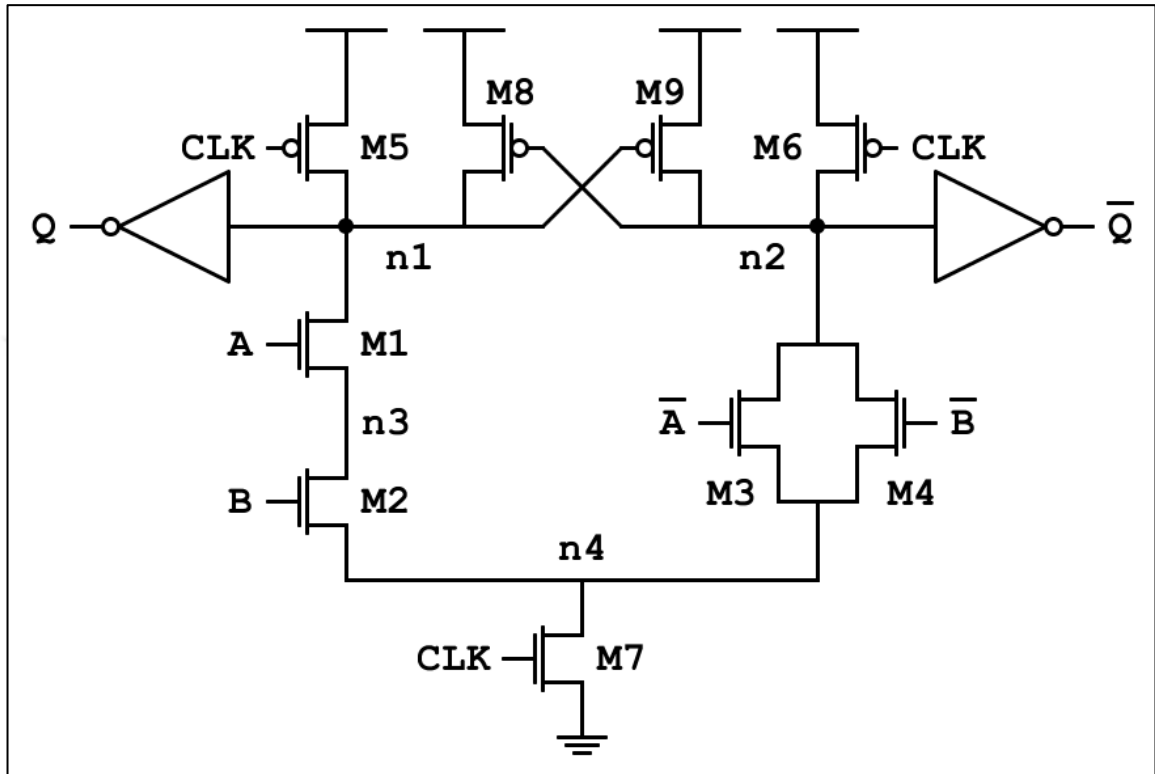


Figure 4.4. A dynamic differential AND gate

4.2.1.1. Internal Structure

Albeit, Figure 4.4 still has a problem: an unbalanced internal structure. Assume that at the end of a precharge phase internal nodes **n1**, **n2**, and **n4** are high, while **n3** is low. Consider that in the next evaluation phase $A = 1$ and $B = 0$. In this case, **n2** and **n4** will be discharged through **M4** and **M7**. Afterwards **M8** will enter conduction mode and keep **n1** high. Since **M2** is in cut-off and **M1** is in conduction, voltage of **n3** will rise through **M1** and **M8** until **M1** goes into cut-off. Afterwards, as inputs and outputs are delayed compared to the clock signal, when **CLK** becomes low previous input combination will persist for a while. Therefore, **n2** will be charged through **M6** and voltage of **n4** will increase because of **M4** and **M6** until **M4** stops conducting. The other nodes will remain high throughout the

precharge phase. Now presume during the next evaluation phase, **A** and **B** are both high. This time **n1**, **n3** and **n4** will be discharged through **M2** and **M7**. Following this, **M8** will go into conduction mode and keep **n2** high. Summary of internal nodes during this sequence is given in Table 4.1.

Figure 4.4 is designed such that **M8** and **M9** are identical, just as **M5** and **M6**. Not to mention, both inverters at the output are made identical as well. However, **n1** only has **M1**, while **n2** has both **M3** and **M4** connected to it from the pull-down network. Thus, **n1** and **n2** do not have very similar equivalent capacitances. It is seen from Table 4.1 that for both input combinations, one of **n1** or **n2** transitions from high to low while the other remains high. For these reasons, power consumption due to these two nodes for different input combinations are as dissimilar as their individual equivalent capacitances. Next, it appears in Table 4.1 that charging and discharging events of **n3** are input dependent. The only node that causes equal power consumption for both input combinations is **n4** in Table 4.1, as it changes from high to low in each case. These imbalances at nodes **n1**, **n2** and **n3** cause vulnerabilities against power analysis attacks.

Table 4.1. Internal nodes of a dynamic differential AND gate during different phases

	n1	n2	n3	n4
Precharge	High	High	Low	High
$A = 1 \quad B = 0$	High	Low	High	Low
Precharge	High	High	High	High
$A = 1 \quad B = 1$	Low	High	Low	Low

4.2.1.2. Sense-Amplifier-Based Logic

The pull-down network of Figure 4.4 should be designed so that, every node except one of **n1** or **n2** is discharged during an evaluation phase and charged during a precharge phase for every input combination. If in addition, **n1** and **n2** have equal total capacitances, total charge

transfer during precharge phases as well as evaluation phases will separately be the same no matter what the inputs are.

One way to create symmetrical $n1$ and $n2$ is through insertion of NMOS transistors between these nodes and the pull-down network. The purpose of is to disconnect these nodes from the pull-down network. Adding two NMOS transistors in this manner forms two cross-coupled inverters with $M8$ and $M9$. This inverter configuration is called a sense-amplifier. Logic style of this kind is named Sense-Amplifier Bases Logic (SABL). To give an example, dynamic differential gate of Figure 4.4 converted to a SABL gate is demonstrated in Figure 4.5.

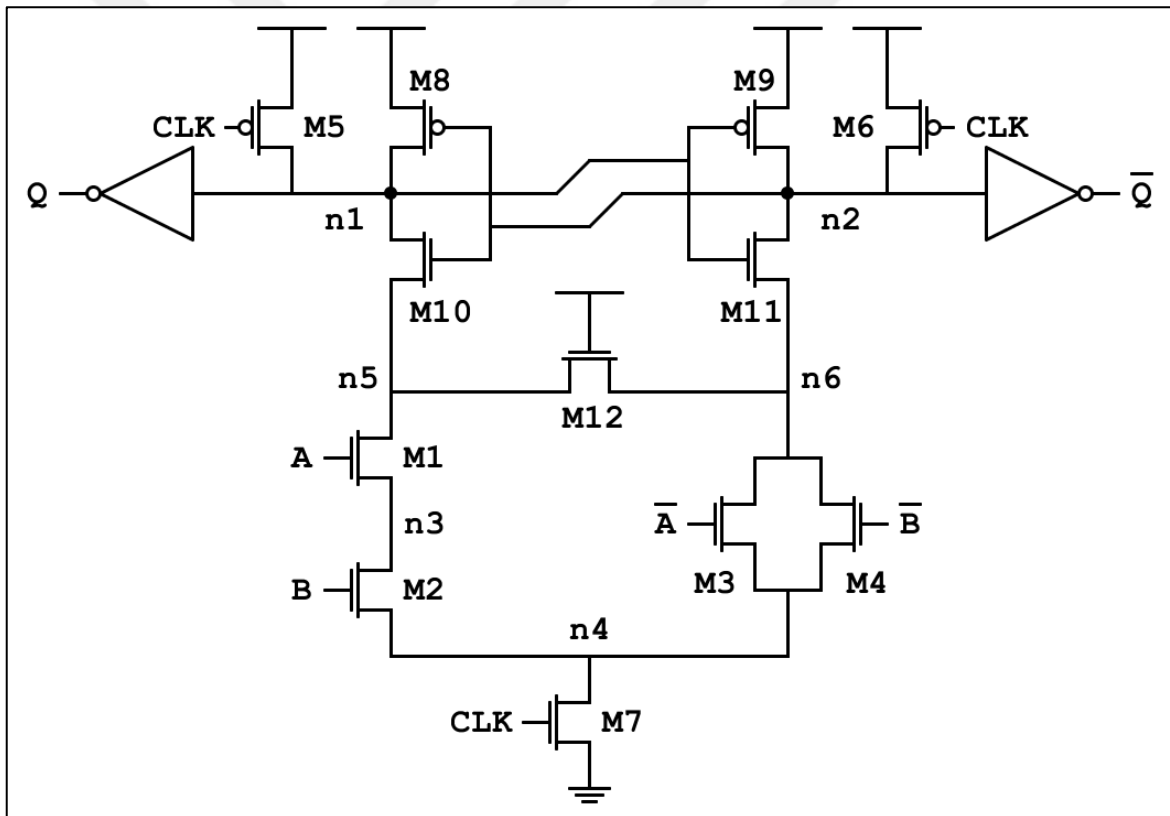


Figure 4.5. Sense-Amplifier-Based Logic for an AND gate

In Figure 4.4, $n1$ is connected to the drain of $M1$, meanwhile $n2$ is connected to the drains of $M3$ and $M4$. This is bound to create imbalance. Yet in Figure 4.5, $n1$ and $n2$ are now connected to identical ports of identical elements: input of an inverter, drains of two PMOS

transistors and drain of an NMOS transistor. Even though one can devise symmetrical pull-down networks for many complementary Boolean functions, it is easier to make **n1** and **n2** very similar while designing the layout, if the pull-down network is taken out of the equation.

M12 in Figure 4.5 is an always conducting transistor with a long channel that acts as a resistance between nodes **n5** and **n6**. After one of **n5** or **n6** is connected to ground, it provides a discharge path for the other. **M12** is utilized for the aim of discharging every node except one of **n1** and **n2** during the evaluation phase. The only time this does not happen is when inputs **A** and **B** are both low. In that situation the pair **M1** and **M2** are in cut-off, meaning there is no low impedance path between **n3** and ground. This can be circumvented by rearranging the pull-down network so that during an evaluation phase, every internal node is connected to either **n4**, **n5** or **n6**.

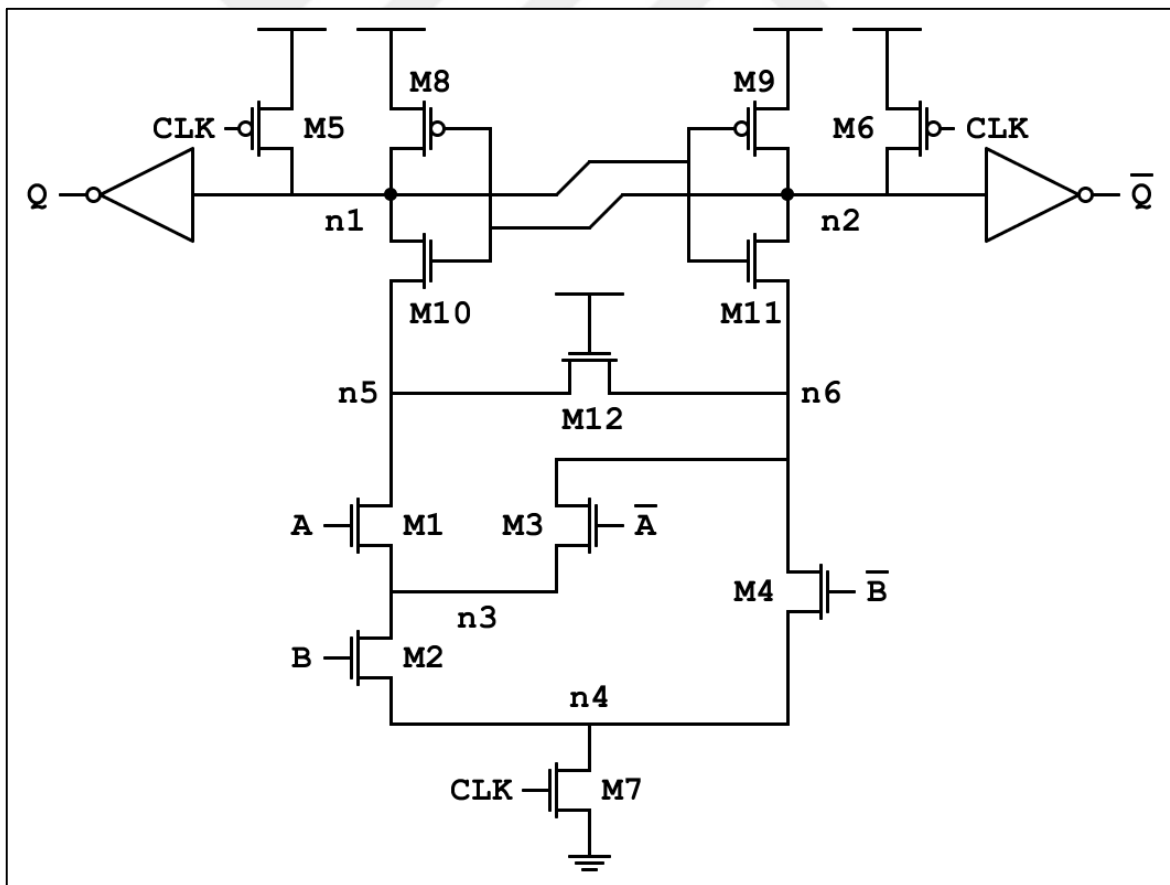


Figure 4.6. SABL with fully connected differential pull-down network for an AND gate

A suggestion is made in [90] by the name of Fully Connected Differential Pull-Down Network. It is a method of transforming two complementary Boolean functions to be more interconnected. Given a Boolean function and its complement, expressions in the form of $x \vee y$ are searched recursively and replaced by the equivalent expression $(x \wedge \bar{y}) \vee y$. If this is done for every logical OR operation until only complementary single literals remain, every node of the pull-down network will be connected to one of **n4**, **n5** or **n6** for every valid input combination. This is demonstrated for the AND gate of Figure 4.5 in Figure 4.6.

As a result of this transformation, when the inputs **A** and **B** are both low in Figure 4.6, **n3** will be discharged, as it is connected to **n6** via **M3**.

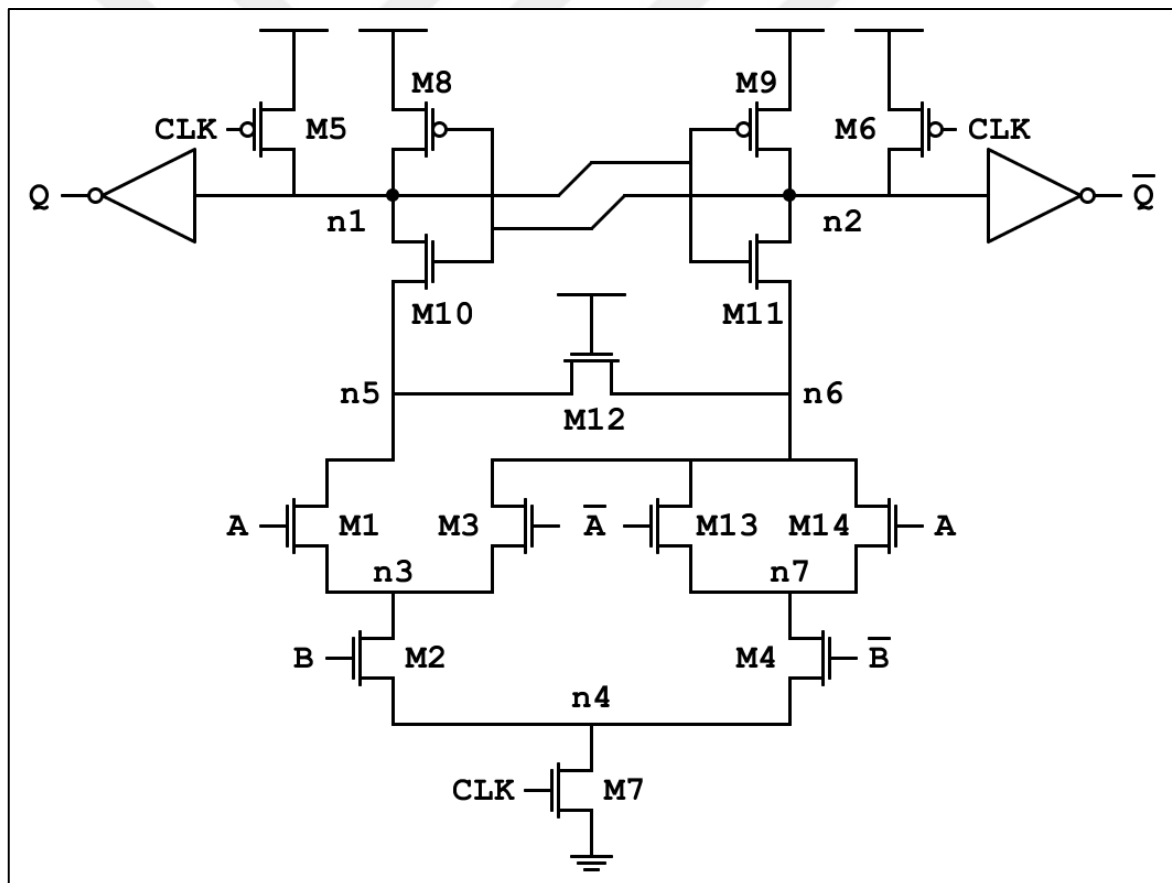


Figure 4.7. SABL with enhanced fully connected differential pull-down network for an AND gate

The paper in [90] makes an addition to this and names it Enhanced Fully Connected Differential Pull-Down Network. In this addition, it attempts to match the resistances of the discharge paths of **n5** and **n6**. To do this, it adds two parallel dummy transistors with complementary inputs one of which will always be in conduction. So-called enhanced version of the gate in Figure 4.6 is displayed in Figure 4.7.

Everything described so far holds for the following SABL XNOR gate in Figure 4.8. XNOR is one of the Boolean functions that actually has size benefits from being built as a differential pull-down network rather than single-ended.

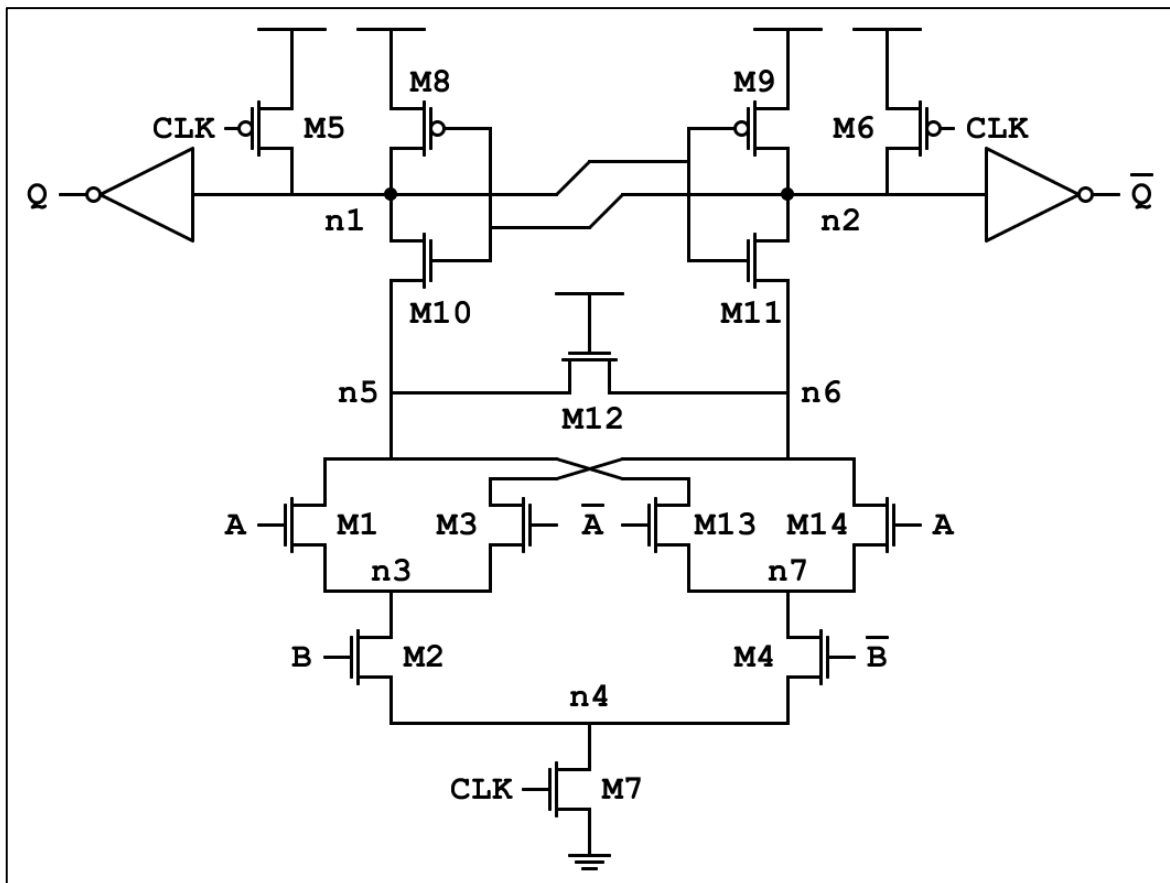


Figure 4.8. SABL with enhanced fully connected differential pull-down network for an XNOR gate

4.2.1.3. Another Dynamic Differential Topology

Although every internal node is charged during the precharge phase of the gate in Figure 4.7, only $n1$ and $n2$ are charged by PMOS switches. Hence, the other nodes are affected by the inefficiencies of using an NMOS as a logical high switch. These inefficiencies include less switching speed compared to a PMOS and inability to conduct beyond logical high voltage minus gate threshold voltage.

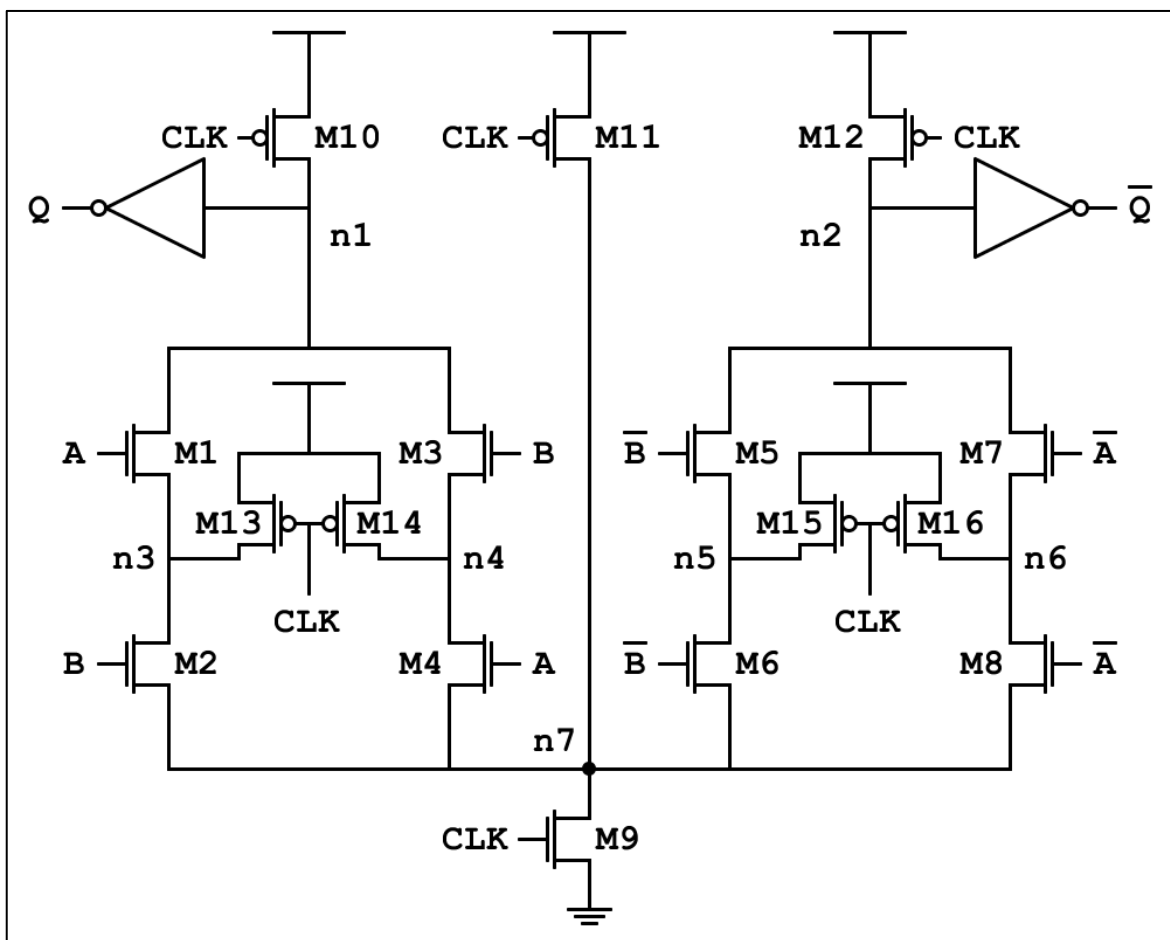


Figure 4.9. An AND gate implemented with ADDT

A solution to this can be found by way of eliminating the sense-amplifier altogether and inserting a PMOS transistor to each internal node as a switch which ties the node to logical high voltage. That way, every node is charged individually by a PMOS. However, these advantages only affect the precharge phase and since the sense-amplifier is removed

discharging of the internal nodes depend on the symmetry of the pull-down network. The pull-down network of Figure 4.7 is not suitable for this. Instead, a fully symmetrical AND pull-down network is used to create the gate in Figure 4.9. This topology will henceforth be called ADDT.

During the evaluation phase of the gate in Figure 4.9, exactly two of $n3$, $n4$, $n5$ and $n6$ of the pull-down network in addition to either $n1$ or $n2$ will be discharged during every input combination. Therefore, if $n3$, $n4$, $n5$ and $n6$ have identical capacitances among themselves, furthermore if the same is true for $n1$ and $n2$, equal amount of current flow will take place for different input combinations.

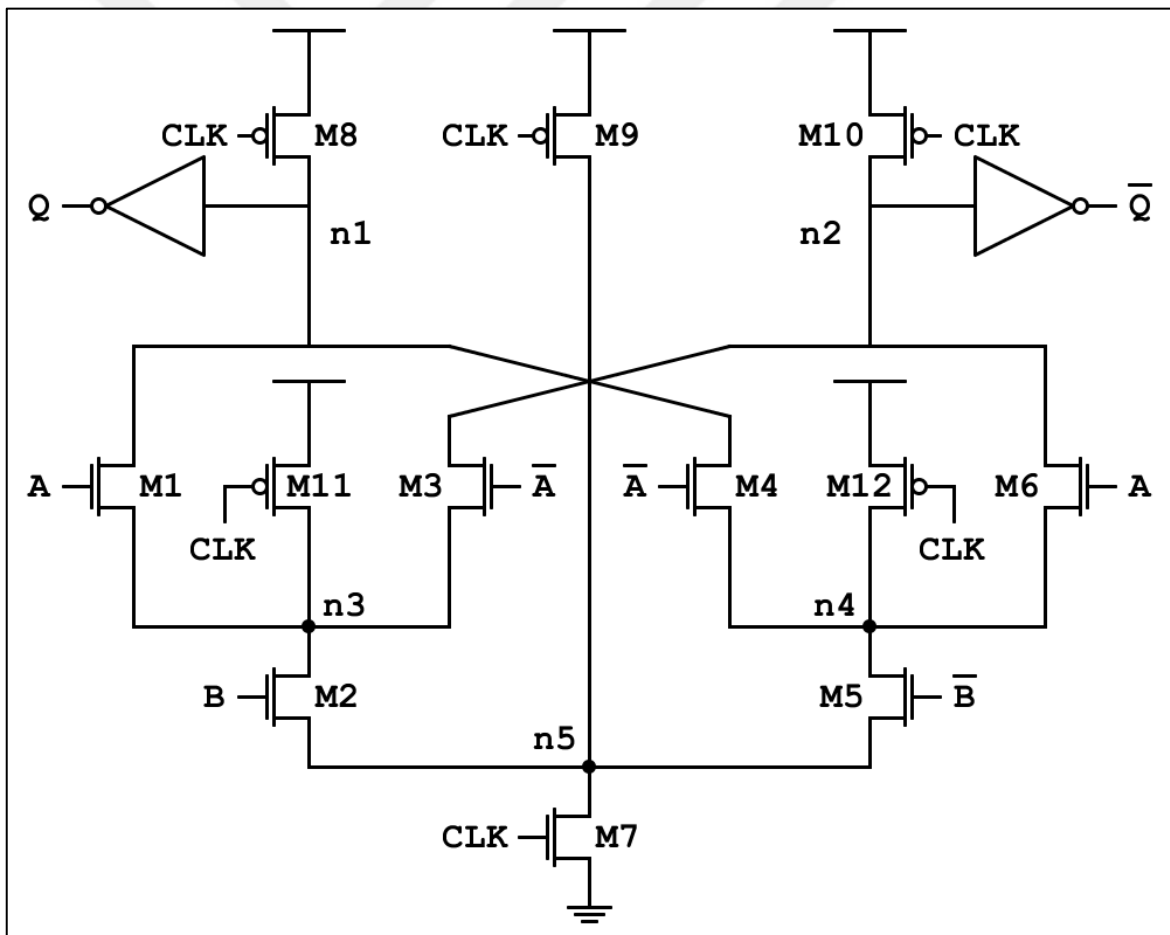


Figure 4.10. An XNOR gate implemented with ADDT

The requirement for symmetry may come at a cost of increased size compared to a SABL gate for some Boolean functions, while it may reduce the size for others. This is demonstrated in Figure 4.10 for an XNOR gate. Besides that, it might be less sound to rely on symmetry of the internal structure, as creating identical devices with identical nodes gets more impractical with bigger and more complex designs.



5. FINITE FIELDS

The purpose of this section is to give a context for how the S-box of AES is implemented using finite field arithmetic. To that effect, it is sought to build a bridge between the definition of a group which is just a set of elements, a binary operation and a couple of rules, and the definitions of a finite field, extension field, normal basis, polynomial basis and eventually the composite fields that are used in all finite field arithmetic implementations of AES S-box. The subject matter is taken from the book written by [106]. The definitions of the book were summarized, rearranged and proven as seen fit.

5.1. GROUPS

A **group** is defined as an arbitrary set with an operation that maps two elements of that set to another element in the same set. If the following rules hold, G is a group together with a binary operation $*$.

- $*$ is associative, i.e. $\forall a, b, c \in G, (a * b) * c = a * (b * c)$.
- G has an identity element e such that $\forall a \in G, e * a = a * e = a$.
- $\forall a \in G$, there exists an inverse $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = e$.

This is enough to call G a group. But additionally, if a group G satisfies $\forall a, b \in G, a * b = b * a$, then it is called a **commutative (abelian) group**.

If $*$ is addition, then the group is called an **additive group**. If it is multiplication, it is named a **multiplicative group**. If every element of an additive group is a multiple of a particular element or every element of a multiplicative group is a power of a particular element, it is called a **cyclic group** and this element is called a **generator** of the group. Every cyclic group is commutative, since an operation between two elements of a cyclic group is basically an operation between a number of copies of the same generator element. If a is a generator of a group G then the group can be symbolized by $G = \langle a \rangle$. A cyclic group can have more than one generator. The rest of the discussion will be about multiplicative groups, but they equally apply to additive groups.

A group is called **finite**, if it has finitely many elements. The number of elements of a finite group is called its **order**.

There is a concept called **equivalence relation** E defined between two elements of a set S . It has three properties by the name of reflexivity, symmetry and transitivity described in order below.

- $\forall a \in S, (a, a) \in E$.
- If $(a, b) \in E$, then $(b, a) \in E$.
- If $(a, b), (b, c) \in E$, then $(a, c) \in E$.

Equivalence relation creates disjoint subsets of S that are called **equivalence classes**. An equivalence class that includes a is designated by $[a]$.

The most common example of an equivalence relation is equality ($=$). Another relevant one is **congruence**. Congruence is equivalence with respect to **modulus** operation i.e. remainder of a division by a quantity. E.g. two numbers a and b are **congruent modulo n** , if $a \equiv b \pmod{n}$.

A group may contain subsets which themselves form a group with respect to the same operation. These are called **subgroups** of the group. The subgroups formed by the identity element and the group itself are called **trivial subgroups**. Every other subgroup is called a **non-trivial subgroup**.

The subset consisting of powers of an element $a \in G$, where G is a multiplicative group, forms a cyclic subgroup of G . Because the fact that a multiplication between any number of powers of a results in another power of a makes the subset a group, and since the subset contains every power of a it is cyclic.

The subgroup formed by set of all powers of a fixed element a is the group generated by a , and is denoted as $\langle a \rangle$. If $a^k = e$, then k is the order of a and any integer m such that $a^m = e$, is a multiple of k . If a is instead a subset S of G , then subgroup generated by finite products of powers of elements of S is signified by $\langle S \rangle$.

For a subgroup H of G , assume an equivalence relation such as $(a, b) \in E$, where $a = b * c$, $\forall a, b \in G$ and $\forall c \in H$. This equivalence relation is called left congruence *modulo* H and

the equivalence classes it partitions G into are called left **cosets** of G modulo H . They are designated by $\mathbf{a} * \mathbf{H} = \{a * h : h \in H\}$. Right cosets would be defined exactly the same except for the order of operations. Consequently, right and left cosets created by a subgroup of a commutative group would be equal.

Number of elements in each left (right) coset of a group modulo one of its subgroups is equal to the order of that subgroup. The reasoning behind this is as follows, cosets are equivalence classes defined by the equivalence relation $R(a, b) \rightarrow b = a * h$ for $b, a \in G, h \in H$, where H is a subgroup of G . Each element of a coset is found by applying this equivalence relation to each element h of H for an element $a \in G$. Therefore, each $a \in G$ can be used to create the coset that it belongs in, and the number of elements of that coset is equal to however many $h \in H$ there exists. Applying the equivalence relation between elements of H results in another element of H , hence creating the coset that is equal to H itself.

Every subgroup of a cyclic group is cyclic. Let H be a subgroup of a cyclic multiplicative group generated by a , i.e. $\langle a \rangle$. Since if $a^n \in H$, then $a^{-n} \in H$, for H is a multiplicative group. This means H has at least one positive power of a . Let x be the least positive integer such that $a^x \in H$ and let $a^y \in H$. If y were not divisible by x then it would be $y = kx + r$ where $r < x$ is the remainder. In that case, $a^y(a^{-x})^k = a^r$ and H being a multiplicative group, $a^r \in H$. But as x is the least positive integer exponent of powers of a that are in H , r has to be zero. Therefore, every element of H can be written as powers of a^x and $H = \langle a^x \rangle$. Hence H is cyclic.

Let $x = \gcd(m, k)$ where m is the order of $\langle a \rangle$ and k is an integer. The order of $\langle a^k \rangle$ is the least positive integer n such that $a^{kn} = e$. For a^{kn} to be equal to e , kn should be a multiple of m . As k is a multiple of x , kn would be a multiple of m only if n were a multiple of $\frac{m}{x}$. The least such n is $\frac{m}{x}$. This implies that for an integer k and a cyclic group $\langle a \rangle$ of order m , a^k generates a subgroup of order $\frac{m}{\gcd(m, k)}$.

Let x be a divisor of the order m of $\langle a \rangle$. There exists only one subgroup of order x in $\langle a \rangle$. This subgroup is generated by $a^{m/x}$. Let $\langle a^k \rangle$ be another subgroup of $\langle a \rangle$ with order x . This means $x = \frac{m}{\gcd(k, m)}$ and $\gcd(k, m) = \frac{m}{x}$. Accordingly, k is a multiple of $\frac{m}{x}$ and $a^k \in \langle a^{m/x} \rangle$.

Thus $\langle a^k \rangle$ is a subgroup of $\langle a^{m/x} \rangle$. But since both $\langle a^k \rangle$ and $\langle a^{m/x} \rangle$ have the same order, they are identical.

For a divisor x of order m of $\langle a \rangle$, there are $\phi(x)$ elements that generate the subgroup of order x . The expression $\phi(x)$ is called Euler's function of x , which is the number of integers n where $1 \leq n \leq x$ that are relatively prime (coprime) to x . Given an integer k , for a^k to generate the subgroup of $\langle a \rangle$ with order x , $\gcd(m, k)$ should be equal to $\frac{m}{x}$. This indicates that the number of elements generating previously mentioned subgroup is the number of integers k such that $1 \leq k \leq m$ and $\gcd(k, m) = \frac{m}{x}$ or equivalently number of integers $k = y \frac{m}{x}$ where $1 \leq y \leq x$ and $\gcd(y, x) = 1$. This is equal to $\phi(x)$.

A cyclic multiplicative group $\langle a \rangle$ of order m has $\phi(m)$ elements that are generators of the whole group. Which means there are $\phi(m)$ integers i that satisfy $\langle a^i \rangle = \langle a \rangle$. This is implied in the previous passage when $x = m$ and the third to last passage when $\gcd(m, n) = 1$.

A mapping $f: G \rightarrow H$ of a group G into a group H is called a **homomorphism** of G into H if $f(a * b) = f(a) \cdot f(b)$ holds $\forall a, b \in G$, where $*$ is the operator of G and \cdot is the operator of H . The **kernel** of homomorphism f is the set **ker f** = $\{a \in G: f(a) = e\}$, where e is the identity element of H .

A subgroup H of G is called a **normal subgroup** if $\forall a \in G$ and $\forall b \in H$, $a \times b \times a^{-1} = b$. Every subgroup of a commutative group is a normal subgroup since $a \times b \times a^{-1} = a \times a^{-1} \times b = b$.

Consider an operation for a subgroup H of G such as $(a * H) * (b * H) = (a * b) * H$. If H is a normal subgroup of G , the cosets of G modulo H form a group under the operation described in the last statement. This group is called a **factor (quotient) group** of G modulo H and is expressed by G/H .

The kernel of homomorphism $f: G \rightarrow H$ is a normal subgroup of G since it maps G to the identity element of H , which means $\forall a \in G$, $e \times a \times e^{-1} = a$. Furthermore, H is isomorphic to $G/\ker f$. Conversely, if D is a normal subgroup of G , $g: G \rightarrow G/D$ is a homomorphism with $\ker g = D$.

5.2. RINGS AND FIELDS

A **ring** is a set R together with two binary operations $\times, +$ is defined with following properties

- R is a commutative group with respect to $+$. Therefore, properties of a commutative group must apply to R and $+$.
- \times is associative, i.e. $(a \times b) \times c = a \times (b \times c)$.
- \times is distributive over $+$, i.e. $a \times (b + c) = a \times b + a \times c$.

$+$ and \times in these definitions are not necessarily identical to ordinary addition and multiplication between numbers. For following definitions $\mathbf{0}$ signifies the identity element of addition. $-$ sign will be used to represent additive inverse, which means $\forall a, b \in R, a - b = 0$.

- A ring, for which \times is commutative and has an identity element $e \neq 0$ and has no **zero divisors** (that is $a \times b = 0$ suggests either $a = 0$ or $b = 0$), is called an **integral domain**.
- A ring is called a **division ring** if non-zero elements of it form a group under multiplication.
- A commutative division ring is called a **field**.

A **subring** is defined in an analogous manner to a subgroup; in that it is a subset of a ring which itself is a ring. Similarly, an **ideal** is the counterpart of normal subgroup for rings. If for a subring J of a ring $R, \forall a \in R$ and $\forall b \in J, a \times b \in J$ then J is an ideal of R . If every element of an ideal J of R can be generated by an element $a \in R$ it is called a **principal ideal** and is represented by $J = (\mathbf{a}) = \{ra : r \in R\}$.

As ideals are normal subgroups of the additive group of R , they can divide R into disjoint cosets called **residue classes**. Residue class that $a \in R$ belongs is written as $[a] = a + J$ (what cosets of G modulo H $a * H$ would be if $*$ were $+$ and H were J). Elements $a, b \in R$ are called congruent modulo J ($a \equiv b \pmod{J}$) if they are in the same residue class modulo J .

Comparable to factor groups, there is a definition called **residue class (factor) ring**. A residue class ring is a ring formed by the residue classes of ring R modulo ideal J together

with the operations $(a + J) + (b + J) = (a + b) + J$ and $(a + J)(b + J) = ab + J$ expressed as R/J .

Let the elements of a finite integral domain R be a_1, \dots, a_n . If a non-zero element $b \in R$ were to be multiplied with every element of R , one would acquire n distinct results. Distinctness of these multiplications is shown by pointing out if $b \times a_i = b \times a_j$, then $a_i = a_j$. But since a_1, \dots, a_n are distinct, $b \times a_1, \dots, b \times a_n$ have to be distinct. None of $b \times a_1, \dots, b \times a_n$ would equal 0 except $b \times 0$ since there are no zero divisors. One of the multiplications would be $b \times a_i = e$. Which means a_i is the multiplicative inverse of b . These conditions would hold $\forall b \in R$ and make R a field. This proves that every finite integral domain is a field.

The ring of residue classes of integers modulo a prime number $\mathbb{Z}/(p)$ is an integral domain with finite order. The explanation is in the following statements. It has an identity element $[1]$. As there are no $[a], [b] \in \mathbb{Z}/(p)$ that divide the prime number p and $[a] \times [b] = [a \times b] = 0$ only if $[a \times b]$ is a multiple of p , there are no zero divisors of $\mathbb{Z}/(p)$. Lastly, it is commutative. As a result of these facts, it is a field.

Homomorphism for rings is defined as $f: R \rightarrow S$, if $\forall a, b \in R$, $f(a + b) = f(a) + f(b)$ and $f(a \times b) = f(a) \times f(b)$ hold. For such f , $\ker f$ is an ideal of R and S is isomorphic to the factor ring $R/\ker f$. Moreover, for an ideal J , the mapping $g: R \rightarrow R/J$ is a homomorphism with $\ker g = J$.

Let $\mathcal{F}_p = \{0, 1, \dots, p - 1\}$ be a set of integers for a prime p and $g: \mathbb{Z}/(p) \rightarrow \mathcal{F}_p$ be a homomorphism such that $g([a]) = a$ for $[a] \in \mathbb{Z}/(p)$ and $a \in \mathcal{F}_p$. An \mathcal{F}_p defined in such manner is a finite field and is dubbed the **Galois field** of order p .

For a ring R there exists a positive integer n such that $\forall a \in R$, $a \times n = 0$. The least such n is called the **characteristic** of R . If no positive integer of that sort exists R is said to have 0 characteristic.

An integral domain R with positive characteristic must have prime characteristic. If the characteristic n were not a prime it could be written as a multiplication of two numbers such as $n = a \times b$, so $(a \times b) \times e = 0$. This means either $(a \times e) = 0$ or $(b \times e) = 0$ as an integral domain has no zero divisors. This implies $\forall c \in R$, one of $a \times c = 0$ or $b \times c = 0$

must be true. This invalidates n being the characteristic, since it is not the least positive integer that fits the definition. A finite field is an integral domain, and for it has finitely many elements, one would eventually reach zero by counting consecutively through integer multiples of identity, thence establishing the existence of a positive characteristic. This means a finite field has prime characteristic.

If R is a commutative ring, $\forall a, b \in R, (a + b)^{p^n} = a^{p^n} + b^{p^n}$. This is because coefficients of $(a + b)^p$ are binomial coefficients $\binom{p}{i}$ for $i = 1, 2, \dots, p$ and knowing that binomial coefficients are integers and $\binom{p}{i}$ has p in its numerator which for $i \neq 1, p$ cannot be cancelled out with any term in the denominator for they are all less than p , $\binom{p}{i}$ is a multiple of p for $i \neq 1, p$. Consequently, every term except a^p and b^p becomes zero. If $n = 2$, $(a + b)^{p^2} = ((a + b)^p)^p = (a^p + b^p)^p = a^{p^2} + b^{p^2}$, and by induction it holds $\forall n \in \mathbb{Z}^+$.

An integral domain is said to be a **principal ideal domain**, if every ideal of it is principal. For a principal ideal domain R , residue class ring $R/(a)$ is a field if and only if a is prime.

5.3. POLYNOMIALS

Polynomials are expressions in the form of $a_0 + a_1x + \dots + a_nx^n$, where a_i are called coefficients and x is a variable. If coefficients $a_0, a_1, \dots, a_n \in R$, then it is called a **polynomial over R** and the variable is called an indeterminate over R . Substituting $b \in R$ in place of the indeterminate of a polynomial over R would result in another $c \in R$.

The ring formed by polynomials over R with two operations polynomial addition and multiplication is called the **polynomial ring** over R . It is represented as $R[x]$.

If the coefficient of the indeterminate with the greatest exponent that is the leading coefficient of a polynomial is the identity element, that polynomial is called a **monic polynomial**. The greatest exponent is called the **degree** of the polynomial (symbolized by $\deg(f)$ for a polynomial f). A polynomial with zero degree is called a **constant polynomial**.

For a field F , a polynomial $g \in F[x]$ divides $f \in F[x]$ or f is **divisible** by g , if there exists a polynomial $h \in F[x]$ such that $f = g \times h$. Otherwise the division of f by g gives $f = g \times h + r$, where $\deg(r) < \deg(g)$.

$F[x]$ is a principal ideal domain where every ideal is generated by a uniquely determined monic polynomial. As a proof consider the polynomial of the least degree of an ideal J . If it is divided by its leading coefficient one gets the monic polynomial $g \in J$. $\forall f \in J$, if one divides f by g the result would be $f = a \times g + r$ for $a, r \in F[x]$. Since $ag \in J$, $f - a \times g = r \in J$, $\deg(r) < \deg(g)$ and g is a member of J with the least degree, $r = 0$. Therefore any $f \in J$ of any ideal J of $F[x]$ can be written as $(g) = \{a \times g : a \in F[x]\}$ for a unique monic polynomial g .

For $f_1, \dots, f_n \in F[x]$, there exists a unique monic polynomial g that divides each f_1, \dots, f_n and any other polynomial that also divides each f_1, \dots, f_n , divides g . Such g is called the **greatest common divisor** of f_1, \dots, f_n . If $g = 1$, f_1, \dots, f_n are called **relatively prime**.

A non-constant polynomial $f \in F[x]$ is said to be **irreducible (prime)** over F if it only has constant polynomial divisors.

Any $f \in F[x]$ can be uniquely written as $c \times p_1^{k_1} \times \dots \times p_n^{k_n}$, where $c \in F$, $k_i \in \mathbb{Z}^+$ and p_i are irreducible polynomials over F . This decomposition is called **prime factorization** of f .

The residue class ring $F[x]/(f)$ for $f \in F[x]$ is a field if and only if f is irreducible over F . Because $F[x]$ is a principal ideal domain and a residue class ring $R/(a)$ of a principal ideal domain R is a field only if a is a prime. The structure of $F[x]/(f)$ is such that it consists of residue classes $\{g + (f) : g \in F[x]\}$, where each residue class can be represented by a unique $\{r \in F[x] : \deg(r) < \deg(f)\}$, that is the remainder of the division of g by f . If $F = \mathcal{F}_p$ and $\deg(f) = n$ then the elements of $\mathcal{F}_p[x]/(f)$ are p^n polynomials of degree $< n$.

An element $a \in F$ is said to be a **root (zero)** of a polynomial $f \in F[x]$, if $f(a)$ where a is substituted for the indeterminate x , equals zero. If $a \in F$ is a root of $f \in F[x]$, f is divisible by $(x - a)^k$, where k is the greatest integer the divisibility holds for and is called the **multiplicity** of a . The roots with $k = 1$ are called **simple roots**, whereas the roots with $k > 1$ are called **multiple roots**. Let $\{a_1, a_2, \dots, a_n\} \in F$ be roots of a polynomial $f \in F[x]$ with

multiplicities $\{k_1, k_2, \dots, k_n\} \in \mathbb{Z}$. Then f is divisible by $(x - a_1)^{k_1} \times (x - a_2)^{k_2} \times \dots \times (x - a_n)^{k_n}$.

A root $a \in F$ of $f \in F[x]$ is a multiple root if and only if it is both a root of f and $\frac{df}{dx}$.

If $f \in F[x]$ is an irreducible polynomial of degree $n \geq 2$, it has no root in F . Otherwise for a root $a \in F$, $(x - a)$ divides f . Thus, it is not irreducible.

5.4. MORE ON FIELDS

A subset of a field that is itself a field is called a **subfield**. If the subfield is not equal to the field that it is a subfield of, then it is named a **proper subfield**. A field is thought of as an **extension field** of its proper subfields.

A subfield of \mathcal{F}_p has to contain 0 and 1 for it to be a field. It also has to contain every other element of \mathcal{F}_p because considering a subfield of \mathcal{F}_p that contains 0 and 1, adding an arbitrary number of 1s together must result in another member of \mathcal{F}_p for that subfield to be a subfield of \mathcal{F}_p . This is a consequence of closure property of a subfield under addition. Therefore \mathcal{F}_p does not possess proper subfields. A field possessing no proper subfields is dubbed a **prime subfield**.

Let K be a subfield and M a subset of F . In this case the field that is the intersection of every field containing both K and M , in other words smallest field consisting of both K and M is called the **extension field** of K attained by adjoining the elements of M and is denoted by $K(M)$. If M consists of a single element θ , then $K(\theta)$ is a **simple extension** with θ being its **defining element**.

If there exists a polynomial $a_0 + a_1\theta + a_2\theta^2 + \dots + a_n\theta^n = 0$ where $\theta \in F$, $a_i \in K$ and not all $a_i = 0$ for K a subfield of F and $i = 1, 2, \dots, n$, θ is said to be **algebraic** over K .

Consider an ideal of $K[x]$ such that $J = \{f \in K[x]: f(\theta) = 0\}$ in which θ is algebraic over K . J is the principal ideal (g) , where g is the monic polynomial of the least degree that satisfies $g(\theta) = 0$. Such a uniquely defined g is called the **minimal polynomial** of θ over

K . g that is described this way is irreducible in K and any $f \in K[x]$ for which $f(\theta) = 0$ is a multiple of g .

An extension field L of K can be thought as a vector space over K . There are a couple of reasons for this comparison. First reason is that the elements of L form an abelian group under addition. Additionally, there are certain distributive and associative laws for multiplication by scalars, namely $(a + b)(\alpha + \beta) = a\alpha + a\beta + b\alpha + b\beta$ and $a(b\alpha) = (ab)\alpha$ where $a\alpha, a\beta, b\alpha, b\beta, ab\alpha \in L, \forall a, b \in K$ and $\forall \alpha, \beta \in L$.

If the extension field L of K described above has finite number of dimensions it is called a **finite extension** and the number of dimensions of the vector space is then called the degree of L over K and is represented by $[L: K]$.

If an extension field L over K has a degree n , then $\forall \theta \in L, \{1, \theta, \theta^2, \dots, \theta^n\}$ a set of n different elements at most, must be linearly dependent since L acts as a vector space over K . Thus, every finite extension of a field is algebraic over that field.

Let g be the minimal polynomial of θ over K of degree n . Furthermore, let $\tau: K[x] \rightarrow K(\theta)$ be a mapping from $K[x]$ to $K(\theta)$. τ is a homomorphism considering $K(\theta)$ is set of polynomials in θ over K , so it maps x to θ and $\ker \tau = \{f(x) \in K[x]: f(\theta) = 0\} = (g)$. This leads to the fact that $K(\theta)$ is isomorphic to $K[x]/(g)$, or equivalently $\varphi: K[x]/(g) \rightarrow K(\theta)$ is an isomorphism because $\vartheta: K[x] \rightarrow K[x]/(g)$ is also a homomorphism with $\ker \vartheta = (g)$.

In conjunction with the definition of g above, n elements $\{1, \theta, \theta^2, \dots, \theta^n\}$ are linearly independent hence they form a basis for the extension $K(\theta)$ over K .

Let g be an irreducible polynomial over K , and $L = K[x]/(g)$. Every element of L is a residue class in the form of $[f] = f + (g)$. Since f is a polynomial in $K[x]$, its residue class can be written as $[f] = [a_0 + a_1x + a_2x^2 + \dots + a_mx^m] = [a_0] + [a_1][x] + [a_2][x]^2 + \dots + [a_m][x]^m$. Furthermore, as for an $a \in K, [a] \rightarrow a$ is an isomorphism and K is a subfield of L , the equation takes the form $a_0 + a_1[x] + a_2[x]^2 + \dots + a_m[x]^m$. This means every element of L may be thought as a polynomial in $[x]$ over K , so L is a simple extension of K with the defining element $[x]$. Lastly, if $g = b_0 + b_1x + b_2x^2 + \dots + b_nx^n$, then $g([x]) = b_0 + b_1[x] + b_2[x]^2 + \dots + b_n[x]^n = [b_0 + b_1x + b_2x^2 + \dots + b_nx^n] = [g] =$

$[0] = 0$. Therefore $[x]$ is a root of g and the following deduction can be made. For an irreducible polynomial over a finite field, there exists a simple extension of that field with the root of the polynomial as a defining element.

Let α and β be roots of an irreducible polynomial f over K . Then, simple extensions $K(\alpha)$ and $K(\beta)$ are isomorphic where $K(\alpha) \rightarrow K(\beta)$ is equivalent to $\alpha \rightarrow \beta$.

Let f be a polynomial of n th degree over a field K . If f can be written as linear products in x as $a(x - \alpha_1)(x - \alpha_2) \dots (x - \alpha_n)$ where a is the leading coefficient of f and $\alpha_1, \alpha_2, \dots, \alpha_n \in F$ are n roots of f , additionally if F is a field obtained by adjoining $\alpha_1, \alpha_2, \dots, \alpha_n$ to K , F is called the splitting field of f over K . F is the extension field $K(\alpha_1, \alpha_2, \dots, \alpha_n)$ and there is no subfield of F that can split f . One can speak of the splitting field of any polynomial over any field. A polynomial f over a field K can be reduced until it is irreducible, then the irreducible parts can be split with adding elements that are not part of K . Ultimately, the smallest field that contains every root of f is the splitting field of f over K . Different splitting fields of f over K are isomorphic under an isomorphism that maps the roots of f that are not elements of K to each other.

A finite field has p^n elements where p is a prime and n is an integer. Since a finite field has a prime characteristic, a prime subfield with that many elements and because it can be represented as a vector space over its prime subfield with integer number of dimensions it has p^n elements, p being its characteristic and n being its degree over its prime subfield.

For a finite field F of q elements, $\forall a \in F, a^q = a$. If $a = 0$, then $a^q = 0$. For other cases, as nonzero elements of F form a multiplicative group of order $q - 1$, $a^{q-1} = 1$ and $a^q = a$.

Let F be a finite field with q elements and K be its subfield. The polynomial $x^q - x$ over K factors in F as $\prod_{a \in F} (x - a)$, so F is the splitting field of $x^q - x$ over K . Since $x^q - x$ and $qx^{q-1} - 1$ has no common roots which in turn means $x^q - x$ has no repeating roots, and F is a finite field with q elements and all q elements of F satisfy $x^q - x$, the smallest finite field that factors $x^q - x$ as such is F . Any finite field with q elements is isomorphic to the splitting field of $x^q - x$ over F_p , the finite field with p elements where p is a prime, hence q is a power of prime p .

Let F be the splitting field of $x^q - x$ over F_p . $S = \{a \in F: a^q - a = 0\}$ is a subfield of F as $0, 1 \in S$ and furthermore, $\forall a, b \in S$, $(a - b)^q = a^q - b^q = a - b$ and $(ab^{-1})^q = a^q(b^{-1})^q = ab^{-1}$ where $b \neq 0$, so $a - b, ab^{-1} \in S$. A finite field defined as such must split $F_p[x] = x^q - x$, since it contains all of its roots. Therefore, $S = F$ and it has q elements because $\forall a \in F$, $a^q = a$. The finite field or equivalently Galois field with q elements is shown by F_q .

F_q where $q = p^n$, p being a prime and n being an integer, only has subfields that have p^m elements, where m is a divisor of n . If an extension field over a finite field with p^m elements has a degree k , it has $(p^m)^k$ elements. Since every finite field that has p^n elements for $n > 1$ is an extension field and vice versa, every finite field only has subfields with the described number of elements.

If m is a divisor of n , $p^m - 1$ divides $p^n - 1$. Because for $n = k \times m$, $p^n - 1 = p^{km} - 1 = (p^m - 1)(p^{(k-1)m} + p^{(k-2)m} + \dots + p^{(k-k)m})$. Then $x^{p^{m-1}} - 1$ divides $x^{p^{n-1}} - 1$ for the same reason. This means $x^{p^m} - x$ divides $x^{p^n} - x$. Therefore $x^{p^n} - x$ contains every root of $x^{p^m} - x$ and the splitting field of $x^{p^n} - x$ has a subfield that is the splitting field of $x^{p^m} - x$.

For a finite group F_q , multiplicative group formed by the non-zero elements of F_q (F_q^*) is cyclic. Let the order of this group be $m = q - 1 = p_1^{r_1} p_2^{r_2} \dots p_n^{r_n}$ where p_i are prime numbers and $r_i \in \mathbb{Z}^+$. Let a_i be an element of F_q^* that is not a root of $x^{p_i} - 1$, and $b_i = a_i^{\frac{m}{p_i}}$ be a set for $i = 1, 2, \dots, n$. $b_i^{p_i^{r_i}} = a_i^m = 1$, so, the order of b_i is a divisor of $p_i^{r_i}$, but since p_i is a prime number, it has to be a power of p_i . Additionally, because a_i is not a root of $x^{p_i} - 1$, which means a_i is an element of a group that is bigger than $F_{\frac{m}{p_i}+1}^*$, hence $a_i^{\frac{m}{p_i}} \neq 1$, the order of b_i is $p_i^{r_i}$. In accordance with these definitions, the order of the element $b = b_1 b_2 \dots b_n$ is m . If it were not so, then its order would be a divisor of m , namely $\frac{m}{p_i}$. This would mean $b^{\frac{m}{p_i}} = b_1^{\frac{m}{p_i}} b_2^{\frac{m}{p_i}} \dots b_n^{\frac{m}{p_i}} = 1$. However, considering $p_j^{r_j}$ divides $\frac{m}{p_i}$ for $i \neq j$, $b_i^{\frac{m}{p_i}} = 1$ for $i \neq j$. Therefore, every term in $b^{\frac{m}{p_i}}$ is 1 except $b_i^{\frac{m}{p_i}}$. Yet $b_i^{\frac{m}{p_i}}$ has to be equal to 1 as well for $b^{\frac{m}{p_i}} = 1$,

and this is impossible because $\frac{m}{p_i} = p_1^{r_1} p_2^{r_2} \dots p_i^{r_i-1} \dots p_n^{r_n}$ is not a multiple of $p_i^{r_i}$, the order of the element b_i . These prove that there is an element $b \in F_q^*$ with order m , that is the generator of the cyclic group F_q^* .

A generator of F_q^* is called a **primitive element** of F_q , and F_q has $\phi(q-1)$ primitive elements.

If F_a is a finite extension of a finite field F_q , then F_a is a simple algebraic extension of F_q with any primitive element of F_a as the defining element. Let α be a primitive element of F_a . Since $F_q(\alpha)$ contains zero and every power of α , $F_a = F_q(\alpha)$.

If F_a is an extension of F_q of order q^n , then its degree over F_q is n , and since $F_a = F_q(\alpha)$ for some primitive element α of F_a , there exists an irreducible polynomial $F_q[x]$ of degree n that is the minimal polynomial of α .

Let $f \in F_q[x]$ be irreducible and $\alpha \in F_{q^n}$ be a root of f . In this case for a $g \in F_q[x]$, $g(\alpha) = 0$ if and only if f divides g . If f is divided by its leading coefficient, it becomes a monic polynomial that is the minimal polynomial of α . The rest follows from the properties of the minimal polynomial.

An irreducible polynomial f of degree m over F_q divides $x^{q^n} - x$ if and only if m divides n . Because if f divides $x^{q^n} - x$, for α a root of f , $\alpha^{q^n} = \alpha$. So $F_q(\alpha)$ is a subfield of F_{q^n} , and as $[F_q(\alpha):F_q] = m$, $F_{q^m} = F_q(\alpha)$ and m divides n . Starting from the assumption that m divides n would imply F_{q^m} is a subfield of F_{q^n} , and if α is a root of f in the splitting field of f over F_q , then $[F_q(\alpha):F_q] = m$, $F_q(\alpha) = F_{q^m}$ and $\alpha \in F_{q^n}$, ergo $\alpha^{q^n} = \alpha$. Consequently, if α is a root of $x^{q^n} - x$, it has to be divisible by f .

Let f be an irreducible polynomial over F_q of degree m , then f splits in F_{q^m} with roots in the form of $\{\alpha, \alpha^q, \alpha^{q^2}, \dots, \alpha^{q^{m-1}}\}$. Take $f(x) = F_q[x] = a_0 + a_1x + \dots + a_mx^m$. In this case $f^q(x) = (a_0 + a_1x + \dots + a_mx^m)^q$. Furthermore, as q is a power of a prime which is equal to the characteristic of F_q , $\forall a \in F_q, qa = 0$. Also considering in $(u+v)^q$, every term has a binomial coefficient $\binom{q}{i}$ that is a multiple of q , except the first and the last one for which the coefficient is equal to one, suggests $f^q(x) = (a_0 + a_1x + \dots + a_mx^m)^q = a_0^q +$

$a_1^q x^q + \dots + a_m^q x^{mq}$. Lastly, because $\forall a \in F_q, a^q = a$, it holds that $f^q(x) = a_0 + a_1 x^q + \dots + a_m x^{mq} = f(x^q)$. Therefore, if $f(\alpha) = 0$, then $f(\alpha^q) = f^q(\alpha) = 0$.

Irreducible polynomials of the same degree over the same finite field has isomorphic splitting fields.

$\forall \alpha \in F_{q^m}$, the n distinct elements $\alpha, \alpha^q, \alpha^{q^2}, \dots, \alpha^{q^{n-1}} \in F_{q^m}$ are named **conjugates** of α with respect to F_q . The number n is the degree of the minimal polynomial of α over F_q and it is a divisor of m . The conjugates of $\alpha \in F_{q^m}^*$ each have the same order, because $F_{q^m}^*$ is cyclic and the order x of α is a divisor of $q^m - 1$, so q^k being a divisor of q^m where $0 < k < m$, the order of α^{q^k} is $\frac{x}{\gcd(x, q^k)} = x$.

For $\alpha \in F_{q^m}$, the **trace** of α over F_q is defined as: $Tr_{F_{q^m}/F_q}(\alpha) = \alpha + \alpha^q + \alpha^{q^2} + \dots + \alpha^{q^{m-1}}$. Following this definition, **discriminant** of $\alpha_1, \alpha_2, \dots, \alpha_m \in F_{q^m}$ is given below in (5.1).

$$\Delta_{F_{q^m}/F_q}(\alpha_1, \alpha_2, \dots, \alpha_m) = \begin{vmatrix} Tr_{F_{q^m}/F_q}(\alpha_1 \alpha_1) & \dots & Tr_{F_{q^m}/F_q}(\alpha_1 \alpha_m) \\ \vdots & \ddots & \vdots \\ Tr_{F_{q^m}/F_q}(\alpha_m \alpha_1) & \dots & Tr_{F_{q^m}/F_q}(\alpha_m \alpha_m) \end{vmatrix} \quad (5.1)$$

$\{\alpha_1, \alpha_2, \dots, \alpha_m\}$ is a base of F_{q^m} over F_q , if and only if $\Delta_{F_{q^m}/F_q}(\alpha_1, \alpha_2, \dots, \alpha_m) \neq 0$. A finite field F_{q^m} has many unique bases over F_q . Nevertheless, certain two types of them have special importance on account of being used widely. These are the **polynomial basis** $\{1, \alpha, \alpha^2, \dots, \alpha^{m-1}\}$ and the **normal basis** $\{\alpha, \alpha^q, \alpha^{q^2}, \dots, \alpha^{q^{m-1}}\}$. Of course, in both cases, the degree of the minimal polynomial of α should be m .

6. ADVANCED ENCRYPTION STANDARD

Advanced Encryption Standard is a symmetrical-key cryptographic algorithm. It has 128-bit, 192-bit and 256-bit variants. The names of these variants indicate key lengths. 128-bit variant has 10 rounds, whereas 192-bit and 256-bit variants have 12 and 14 rounds respectively.

There are five main operations performed. These are called `KeyExpansion`, `AddRoundKey`, `SubBytes`, `ShiftRows` and `MixColumns`. The latter four are performed on what is called a state matrix. State matrix is initially formed by splitting the plaintext into its bytes and putting it in a 4-by- x matrix. The value of x depends on the variant. The key as well is put into a matrix of same size. The state matrix progressively gets modified by previously mentioned four operations, meanwhile the key matrix is updated every round by `KeyExpansion` operation. `AddRoundKey` is an element-wise XOR operation between the key matrix and the state matrix. `SubBytes` is a substitution operation applied to each byte separately. `ShiftRows` and `MixColumns` can be jointly thought as permutation operations applied to the matrix as a whole. Having described the operations, below is how they are implemented:

- `KeyExpansion` derives as many keys as there are rounds in addition to the original key.
- `AddRoundKey` is executed with the original key and plaintext as its inputs.
- `SubBytes`, `ShiftRows`, `MixColumns` and `AddRoundKey` are applied to the state matrix in written order for 9, 11 or 13 times utilizing another key each round.
- One last round of `SubBytes`, `ShiftRows` and `AddRoundKey` sequence is run to obtain the ciphertext.

6.1. SUBBYTES

The `SubBytes` operation bears great importance both for AES algorithm and power analysis. The importance stems from its non-linearity. While this aids to inconvenience

attacks based on mathematical methods that focus on the algorithm itself, it inadvertently makes side-channel attacks based on power consumption easier.

This non-linearity is accomplished via a multiplicative inversion in a finite field followed by matrix multiplication and bit inversion. The multiplicative inversion is done in a Galois field of order 2^8 (also shown by $GF(2^8)$) modulo the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$.

`SubBytes` is applied to the plaintext byte-wise. For that reason, a byte of plaintext is thought of as an element in $GF(2^8)$. This works because $GF(2^8)$ has 2^8 elements and it can be represented as a vector space over $GF(2)$ which has two elements: zero and one.

6.1.1. Multiplicative Inversion in $GF(2^8)$

A straightforward inversion in $GF(2^8)$ is difficult. Instead, the inversion of the whole substitution can be done using a look-up table. The case of calculating the whole `SubBytes` by a look-up table would be the fastest case in terms of delay time of the digital circuitry. But it would also require two different 2^8 -byte look-up tables, one for each of encryption and decryption. It would also be unsuitable for pipelining and necessitate the use of additional tables for each parallel `SubBytes` operation. The instantaneous power consumption would be high, which would make it particularly vulnerable to power analysis attacks. The case where a look-up table is used just for inversion would only require a single 2^8 -byte look-up table and the logic circuit for the affine transformations. However, it would be just as vulnerable to power analysis attacks because of the similar instantaneous power consumption.

Another method is to calculate the inversion in the isomorphic fields such as the extension field $GF((2^4)^2)$ of $GF(2^4)$ instead of $GF(2^8)$, which is an extension over $GF(2)$. Finding the inverse in $GF((2^4)^2)$ is easier owing to the polynomials (elements of the field) being first-degree modulo a second-degree irreducible polynomial over $GF(2^4)$. In order to demonstrate, let $A = a_1x + a_0$ be an element of $GF((2^4)^2)$, and $A^{-1} = a'_1x + a'_0$ be its inverse. It holds that $A \times A^{-1} = 1$, which corresponds to the expression in (6.1).

$$(a_1x + a_0)(a'_1x + a'_0) = 1 \quad (6.1)$$

The multiplication is done modulo the irreducible polynomial $x^2 + b_1x + b_0$. Hence, the right-hand side of the multiplication in (6.1) equals the remainder of $a_1a'_1x^2 + (a_1a'_0 + a_0a'_1)x + a_0a'_0$ divided by $x^2 + b_1x + b_0$, which is given by (6.2).

$$\begin{aligned} & (a_1a'_0 + a_0a'_1 - a_1a'_1b_1)x + a_0a'_0 - a_1a'_1b_0 \\ & = (a_1a'_0 + a_0a'_1 + a_1a'_1b_1)x + a_0a'_0 + a_1a'_1b_0 \end{aligned} \quad (6.2)$$

as additive inverse (negative) of an element in $GF(2^n)$ is equal to itself for $n \in \mathbb{Z}^+$. This is because the characteristic of these Galois fields is equal to two. All in all, left-hand side of (6.1) equals (6.2) and a'_1 and a'_0 is deduced as in (6.3) where α is given in (6.4).

$$\begin{aligned} & (a_1a'_0 + a_0a'_1 + a_1a'_1b_1)x + a_0a'_0 + a_1a'_1b_0 = 1 \\ & (a_1a'_0 + a_0a'_1 + a_1a'_1b_1) = 0 \quad a_0a'_0 + a_1a'_1b_0 = 1 \\ & a'_1 = \alpha^{-1}a_1 \quad a'_0 = \alpha^{-1}(a_0 + a_1b_1) \end{aligned} \quad (6.3)$$

$$\alpha = a_0^2 + a_1a_0b_1 + a_1^2b_0 \quad (6.4)$$

where $a_1, a_0, a'_1, a'_0, b_1, b_0 \in GF(2^4)$. Therefore, the inversion in $GF(2^8)$ is reduced to inversion, multiplication and addition in $GF(2^4)$.

Addition in $GF(2^n)$ such that $n \in \mathbb{Z}^+$, is just addition of polynomials with binary coefficients (coefficients in $GF(2)$). It can be realized in a digital circuit by bit-wise logical XOR of coefficients of indeterminates with the same exponent. When it comes to the inversion and multiplication in $GF(2^4)$, they can be accomplished by either 16-byte look-up tables, simplification of the Boolean functions that are directly derived from multiplication and inversion in $GF(2^4)$, or a reduction process identical to what has been done for $GF(2^8)$, i.e. calculating the multiplication and multiplicative inverse in $GF((2^2)^2)$. The last method is completely analogous to the corresponding method of the inversion in $GF((2^4)^2)$. As a result, the multiplication $C \times C^{-1} = 1$ for $C = c_1x + c_0$ and its inverse $C^{-1} = c'_1x + c'_0$ in $GF((2^2)^2)$ modulo $x^2 + d_1x + d_0$ is equal to the expression in (6.5).

$$(c_1c'_0 + c_0c'_1 + c_1c'_1d_1)x + c_0c'_0 + c_1c'_1d_0 \quad (6.5)$$

Moreover, the coefficients of the multiplicative inverse C^{-1} can be found as $c'_1 = \beta^{-1}c_1$ and $c'_0 = \beta^{-1}(c_0 + c_1d_1)$, where $\beta = c_0^2 + c_1c_0d_1 + c_1^2d_0$ and $c_1, c_0, c'_1, c'_0, d_1, d_0 \in GF(2^2)$.

It follows henceforth that the inversion and multiplication in $GF(2^2)$ is achieved exactly in the same manner. The only irreducible polynomial over $GF(2)$ of second degree is $x^2 + x + 1$. Therefore, operations in $GF(2^2)$ are carried out modulo $x^2 + x + 1$. This leads to the following expressions for the inverse and multiplication in $GF(2^2)$. If $E = e_1x + e_0$ is an element of $GF(2^2)$ and $E^{-1} = e'_1x + e'_0$ is its inverse, then e'_1 and e'_0 can be found as in (6.6).

$$\begin{aligned}
E \times E^{-1} &= (e_1x + e_0)(e'_1x + e'_0) = (e_1e'_0 + e_0e'_1 + e_1e'_1)x + e_0e'_0 + e_1e'_1 \\
&= 1 \\
e'_1 &= (e_0^2 + e_1e_0 + e_1^2)^{-1}e_1 = (e_0 + e_1e_0 + e_1)e_1 = e_1e_0 + e_1^2e_0 + e_1^2 \\
&= e_1 \\
e'_0 &= (e_0^2 + e_1e_0 + e_1^2)^{-1}(e_0 + e_1) = (e_0 + e_1e_0 + e_1)(e_0 + e_1) \\
&= e_0^2 + e_1e_0^2 + e_1e_0 + e_1e_0 + e_1^2e_0 + e_1^2 = e_0 + e_1
\end{aligned} \tag{6.6}$$

for $e_1, e_0, e'_1, e'_0 \in GF(2) = \{0,1\}$. The simplifications in (6.6) are due to the identities $e^2 = e^{-1} = e$ and $e + e = 0$ for $e \in GF(2)$.

In $GF(2)$, operations can easily be implemented by simple logic gates. Addition and multiplication in $GF(2)$ are basically logical XOR and logical AND. Through the reduction steps explained in the previous passages, multiplicative inverse of an element in $GF(2^8)$ can be calculated from bottom-up by doing the arithmetic in $GF(2)$ and hierarchically going all the way up to $GF(((2^2)^2)^2)$.

All the previous discussion was made for polynomial bases. This leaves the choice of which root of an irreducible polynomial defined over a finite field being extended to use, while constructing a simple extension. There are two choices for roots of irreducible polynomials of second degree for each of $GF(2^2)$, $GF((2^2)^2)$ and $GF(((2^2)^2)^2)$. Another option is to use normal basis to represent $GF(2^2)$, $GF((2^2)^2)$ and $GF(((2^2)^2)^2)$, which uses every root of an irreducible polynomial as a basis for an extension.

Everything calculated so far for polynomial bases can be done for normal bases as well. In accordance with the previous derivations, let $A = a_1x_1 + a_0x_0$ and its inverse $A^{-1} = a'_1x_1 + a'_0x_0$ be elements of $GF((2^4)^2)$ with $x_1, x_0 \in GF((2^4)^2)$ being two roots of $x^2 +$

$b_1x + b_0$, where $x_0 = x_1^{2^4}$, $x_1 = x_0^{2^4}$, $b_1 = x_1 + x_0$ and $b_0 = x_1x_0$. The multiplication $A \times A^{-1} = 1$ turns into the equality in (6.7).

$$\begin{aligned} (a_1x_1 + a_0x_0)(a'_1x_1 + a'_0x_0) \\ &= (a_1a'_1)x_1^2 + (a_1a'_0 + a_0a'_1)x_1x_0 + (a_0a'_0)x_0^2 \\ &= (a_1a'_1b_1 + \alpha)x_1 + (a_0a'_0b_1 + \alpha)x_0 = 1 \end{aligned} \quad (6.7)$$

where $\alpha = (a_1 + a_0)(a'_1 + a'_0)b_1^{-1}b_0$. Because $1 = b_1^{-1}(x_1 + x_0)$, one can solve equation (6.7) for a'_1 and a'_0 in terms of a_1 , a_0 , b_1 , and b_0 as given below in (6.8).

$$\begin{aligned} (a_1a'_1b_1 + \alpha)x_1 + (a_0a'_0b_1 + \alpha)x_0 &= b_1^{-1}x_1 + b_1^{-1}x_0 \\ (a_1a'_1b_1 + \alpha) &= b_1^{-1} \quad (a_0a'_0b_1 + \alpha) = b_1^{-1} \\ a'_1 &= \beta^{-1}a_0 \quad a'_0 = \beta^{-1}a_1 \\ \beta &= a_1a_0b_1^2 + (a_1 + a_0)^2b_0 \end{aligned} \quad (6.8)$$

where $a_1, a_0, a'_1, a'_0, b_1, b_0 \in GF(2^4)$. This reduces inversion in $GF(2^8)$ to operations in $GF(2^4)$. Corresponding equations for $GF((2^2)^2)$ and $GF(2^2)$ are provided in (6.9) for $GF((2^2)^2)$ and (6.10) for $GF(2^2)$ after the simplifications for $GF(2)$ are applied as before.

$$\begin{aligned} C &= c_1y_1 + c_0y_0 \quad C^{-1} = c'_1y_1 + c'_0y_0 \\ (C \times C^{-1}) \bmod (y^2 + d_1y + d_0) &= (c_1c'_1d_1 + \gamma)y_1 + (c_0c'_0d_1 + \gamma)y_0 \\ \gamma &= (c_1 + c_0)(c'_1 + c'_0)d_1^{-1}d_0 \\ c'_1 &= \delta^{-1}c_0 \quad c'_0 = \delta^{-1}c_1 \\ \delta &= c_1c_0d_1^2 + (c_1 + c_0)^2d_0 \\ y_0 &= y_1^{2^2} \quad y_1 = y_0^{2^2} \quad d_1 = y_1 + y_0 \quad d_0 = y_1y_0 \\ C, C^{-1}, y_1, y_0 &\in GF((2^2)^2) \quad c_1, c_0, c'_1, c'_0, d_1, d_0 \in GF(2^2) \end{aligned} \quad (6.9)$$

$$\begin{aligned} E &= e_1z_1 + e_0z_0 \quad E^{-1} = e'_1z_1 + e'_0z_0 \\ (E \times E^{-1}) \bmod (z^2 + z + 1) \\ &= (e_1e'_0 + e_0e'_1 + e_0e'_0)z_1 + (e_1e'_1 + e_1e'_0 + e_0e'_1)z_0 \\ e'_1 &= e_0 \quad e'_0 = e_1 \\ z_0 &= z_1^2 \quad z_1 = z_0^2 \quad 1 = z_1 + z_0 \quad 1 = z_1z_0 \\ E, E^{-1}, z_1, z_0 &\in GF(2^2) \quad e_1, e_0, e'_1, e'_0 \in GF(2) \end{aligned} \quad (6.10)$$

The isomorphic fields $GF(((2^2)^2)^2)$ or $GF((2^4)^2)$ used to represent $GF(2^8)$ by one of the methods explained here are called composite fields or tower fields in the literature.

6.1.2. Variations on Logical Functions for SubBytes

Different variations mainly have differences in what basis is used at each level of field extension for the inversion in $GF(2^8)$ and how many levels are utilized. What is meant by levels of extension is how many reductions as explained in the previous subsection are made until the finite field arithmetic operations are actualized with simple logical gates.

If the multiplicative inversion will not be done in $GF(2^8)$ modulo the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$, the element of $GF(2^8)$ to be inverted should be mapped to the new isomorphic field, inverted in that field and then mapped back to the original field in order to conform to the standard of finding the inverse in the original finite field the AES algorithm is designed for. Thereupon, depending on the new isomorphic field the inverse will be found in, two conversion matrices should be employed before and after the inversion. The choice of isomorphic field is also affected by these matrices. As conversion matrices are applied via matrix multiplication, matrices with more zeroes are preferable in order to reduce the logic gates necessary for the multiplication.

Five different SubBytes implementations are considered in this thesis. They vary in their aforementioned properties. They are based on five different academic papers on the subject. For the purposes of this text, they are named after one of the authors' names from the corresponding paper that they are mainly adapted from. Five of them will be called *Walkerstorfer* [30], *Boyar* [54], *Canright* [43], *Nogami* [45] and *Nekado* [46].

All of them except *Boyar* share a common structure. The structure is as such:

1. The input of SubBytes is converted from the original field to a new field by multiplying a mapping matrix T of size (8×8) modulo 2 from left with the input written as a column matrix where the elements of the column matrix are the coefficients of the polynomial which is an element of $GF(2^8)$. The coefficients are ordered in the column matrix such that, as they are written from top to bottom, the exponent of the indeterminate that the coefficient belongs to goes from greatest to least.
2. Multiplicative inverse is found in the new field

3. The inverse is converted back to the original field by multiplying it modulo 2 from left with a mapping matrix that is the matrix inverse of T modulo 2
4. The affine transformation is applied by multiplying the inverse in the original field with A matrix from left and adding B column matrix to the product modulo 2. A and B matrices are shown below in (6.11).

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \quad (6.11)$$

The whole process can be represented by the expression in (6.12).

$$Output_{SubBytes} = A \times T^{-1} \times (T \times Input_{SubBytes})^{-1} + B \quad (6.12)$$

The multiplicative inverse of $(T \times Input_{SubBytes})$ is calculated in $GF(2^8)$. Multiplications, addition and multiplicative inverse of T in (6.12) are calculated modulo 2, meaning that the remainder after division of every element of the resultant matrices by two is taken as the final result. Under these circumstances, multiplications with constant matrices in (6.12) can be realized with XOR gates only, whereas the addition of B matrix is equivalent to merely applying logical NOT to certain bits.

The input and the output are both column matrices that describe polynomials of the form $C = c_7x^7 + c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$ for $C \in GF(2^8)$. C is represented in column matrix form as in (6.13).

$$C = \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} \quad (6.13)$$

6.1.3. Wolkerstorfer

This `SubBytes` implementation calculates the multiplicative inverse in $GF((2^4)^2)$. It uses polynomial bases for both $GF((2^4)^2)$ and $GF(2^4)$. $GF(2^4)$ is defined with the basis $\{x_0^3, x_0^2, x_0, 1\}$ in which x_0 , the defining element, is a root of the irreducible polynomial $x^4 + x + 1$ over $GF(2)$. $GF((2^4)^2)$ is defined with the basis $\{y_0, 1\}$ in which y_0 is a root of the irreducible polynomial $y^2 + y + x_0^3 + x_0^2 + x_0$ over $GF(2^4)$.

The matrix that converts the original $GF(2^8)$ to $GF((2^4)^2)$ (T) and its inverse (T^{-1}) are given by (6.14).

$$T = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.14)$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

An element $D \in GF((2^4)^2)$ is represented as $D = d_1 y_0 + d_0$ for which $d_1, d_0 \in GF(2^4)$. d_1 and d_0 can be written as $d_1 = d_{13}x_0^3 + d_{12}x_0^2 + d_{11}x_0 + d_{10}$ and $d_0 = d_{03}x_0^3 + d_{02}x_0^2 + d_{01}x_0 + d_{00}$ in which $d_{13}, d_{12}, d_{11}, d_{10}, d_{03}, d_{02}, d_{01}, d_{00} \in GF(2)$. All in all, the column matrix representation of an element D transforms into the expression in (6.15).

$$D = \begin{bmatrix} d_{13} \\ d_{12} \\ d_{11} \\ d_{10} \\ d_{03} \\ d_{02} \\ d_{01} \\ d_{00} \end{bmatrix} \quad (6.15)$$

Next, Boolean functions of multiplication and multiplicative inversion in $GF(2^4)$ are needed to fulfill equation (6.3) of finding the multiplicative inverse in $GF((2^4)^2)$. These functions are given in terms of summation and multiplication of two elements in $GF(2)$, which correspond to XOR and AND operations.

Product P of two elements $E, F \in GF(2^4)$ is given by (6.16).

$$\begin{aligned}
 E &= \begin{bmatrix} e_3 \\ e_2 \\ e_1 \\ e_0 \end{bmatrix} & F &= \begin{bmatrix} f_3 \\ f_2 \\ f_1 \\ f_0 \end{bmatrix} & P &= \begin{bmatrix} p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix} \\
 \alpha &= e_0 + e_3 & \beta &= e_2 + e_3 \\
 p_0 &= e_0f_0 + e_3f_1 + e_2f_2 + e_1f_3 & p_1 &= e_1f_0 + \alpha f_1 + \beta f_2 + (e_1 + e_2)f_3 \\
 p_2 &= e_2f_0 + e_1f_1 + \alpha f_2 + \beta f_3 & p_3 &= e_3f_0 + e_2f_1 + e_1f_2 + \alpha f_3
 \end{aligned} \tag{6.16}$$

There are two special multiplications that are more convenient to implement on their own. They are multiplication of an element by itself (squaring) and multiplication by a constant, namely $R = x_0^3 + x_0^2 + x_0$ for a root $x_0 \in GF(2^4)$ of $x^4 + x + 1$ over $GF(2)$. Let $E, S, Q, R \in GF(2^4)$ be given as (6.17).

$$E = \begin{bmatrix} e_3 \\ e_2 \\ e_1 \\ e_0 \end{bmatrix} \quad S = \begin{bmatrix} s_3 \\ s_2 \\ s_1 \\ s_0 \end{bmatrix} \quad Q = \begin{bmatrix} q_3 \\ q_2 \\ q_1 \\ q_0 \end{bmatrix} \quad R = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{6.17}$$

Then $S = E^2$ is given by (6.18).

$$s_0 = e_0 + e_2 \quad s_1 = e_2 \quad s_2 = e_1 + e_3 \quad s_3 = e_3 \tag{6.18}$$

Subsequently, elements q_0, q_1, q_2 and q_3 of $Q = R \times E$ are computed as in (6.19).

$$\begin{aligned}
 \alpha &= e_0 + e_1 & \beta &= e_2 + e_3 \\
 q_0 &= e_1 + \beta & q_1 &= \alpha & q_2 &= \alpha + e_2 & q_3 &= \alpha + \beta
 \end{aligned} \tag{6.19}$$

The multiplicative inverse E^{-1} of $E \in GF(2^4)$ is given by (6.20).

$$\begin{aligned}
E &= \begin{bmatrix} e_3 \\ e_2 \\ e_1 \\ e_0 \end{bmatrix} & E^{-1} &= \begin{bmatrix} e'_3 \\ e'_2 \\ e'_1 \\ e'_0 \end{bmatrix} \\
\alpha &= e_1 + e_2 + e_3 + e_1 e_2 e_3 \\
e'_0 &= \alpha + e_0 + e_0 e_2 + e_1 e_2 + e_0 e_1 e_2 \\
e'_1 &= e_0 e_1 + e_0 e_2 + e_1 e_2 + e_3 + e_1 e_3 + e_0 e_1 e_3 \\
e'_2 &= e_0 e_1 + e_2 + e_0 e_2 + e_3 + e_0 e_3 + e_0 e_2 e_3 \\
e'_3 &= \alpha + e_0 e_3 + e_1 e_3 + e_2 e_3
\end{aligned} \tag{6.20}$$

6.1.4. Boyar

This `SubBytes` implementation expands $GF(2^8)$ to $GF(((2^2)^2)^2)$. It uses the basis $\{x_0^2, x_0\}$ for a root x_0 of the irreducible polynomial $x^2 + x + 1$ over $GF(2)$ to define $GF(2^2)$. Then it uses $\{y_0^4, y_0\}$ in which y_0 is a root of the irreducible polynomial $y^2 + y + x_0$ over $GF(2^2)$ to define $GF((2^2)^2)$. J. Boyar et al. [54] do not further specify which basis or irreducible polynomial is used to represent $GF(((2^2)^2)^2)$.

In the paper written by J. Boyar et al. [54], `SubBytes` is classified into two different parts called linear and non-linear. Linearity is decided here by the existence of logical AND operation. Afterwards, the linear and non-linear sections are subjected to heuristic reduction methods described in the paper, which aims to group and cancel the similar parts for the linear sections and reduce number of AND gates for the non-linear sections. Nonetheless, this project does not resort to the reductions applied to the linear sections in favor of automated ad hoc reductions during logic synthesis.

It is evident at a first glance that (6.12) roughly has a non-linear multiplicative inversion part preceded and succeeded by linear parts that are matrix multiplications. It is in fact what is discovered in the paper written by J. Boyar et al. [54] with a little difference regarding where the different parts start and end.

In conclusion, this implementation brings about `SubBytes` by means of the following equation in (6.21).

$$Output_{SubBytes} = K \times F(U \times Input_{SubBytes}) \tag{6.21}$$

K and U in (6.21) are 8-by-18 and 22-by-8 matrices, while F is a non-linear function with 22-bit input and 18-bit output. They are given by (6.22), (6.23) and (6.24), which leads to the set of operations in (6.25) for output computed with respect to the input.

$$K = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \quad (6.22)$$

$$U = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (6.23)$$

$$F_{input} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{20} \\ y_{21} \end{bmatrix} \quad F_{output} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{16} \\ z_{17} \end{bmatrix} \quad (6.24)$$

$$\begin{aligned}
t_2 &= y_{12} \times y_{15} & t_3 &= y_3 \times y_6 & t_4 &= t_3 + t_2 \\
t_5 &= y_4 \times y_0 & t_6 &= t_5 + t_2 & t_7 &= y_{13} \times y_{16} \\
t_8 &= y_5 \times y_1 & t_9 &= t_8 + t_7 & t_{10} &= y_2 \times y_7 \\
t_{11} &= t_{10} + t_7 & t_{12} &= y_9 \times y_{11} & t_{13} &= y_{14} \times y_{17} \\
t_{14} &= t_{13} + t_{12} & t_{15} &= y_8 \times y_{10} & t_{16} &= t_{15} + t_{12} \\
t_{17} &= t_4 + t_{14} & t_{18} &= t_6 + t_{16} & t_{19} &= t_9 + t_{14} \\
t_{20} &= t_{11} + t_{16} & t_{21} &= t_{17} + y_{20} & t_{22} &= t_{18} + y_{19} \\
t_{23} &= t_{19} + y_{21} & t_{24} &= t_{20} + y_{18} \\
\\
t_{25} &= t_{21} + t_{22} & t_{26} &= t_{21} \times t_{23} & t_{27} &= t_{24} + t_{26} \\
t_{28} &= t_{25} \times t_{27} & t_{29} &= t_{28} + t_{22} & t_{30} &= t_{23} + t_{24} \\
t_{31} &= t_{22} + t_{26} & t_{32} &= t_{31} \times t_{30} & t_{33} &= t_{32} + t_{24} \\
t_{34} &= t_{23} + t_{33} & t_{35} &= t_{27} + t_{33} & t_{36} &= t_{24} \times t_{35} \\
t_{37} &= t_{36} + t_{34} & t_{38} &= t_{27} + t_{36} & t_{39} &= t_{29} \times t_{38} \\
t_{40} &= t_{25} + t_{39} \\
\\
t_{41} &= t_{40} + t_{37} & t_{42} &= t_{29} + t_{33} & t_{43} &= t_{29} + t_{40} \\
t_{44} &= t_{33} + t_{37} & t_{45} &= t_{42} + t_{41} & z_0 &= t_{44} \times y_{15} \\
z_1 &= t_{37} \times y_6 & z_2 &= t_{33} \times y_0 & z_3 &= t_{43} \times y_{16} \\
z_4 &= t_{40} \times y_1 & z_5 &= t_{29} \times y_7 & z_6 &= t_{42} \times y_{11} \\
z_7 &= t_{45} \times y_{17} & z_8 &= t_{41} \times y_{10} & z_9 &= t_{44} \times y_{12} \\
z_{10} &= t_{37} \times y_3 & z_{11} &= t_{33} \times y_4 & z_{12} &= t_{43} \times y_{13} \\
z_{13} &= t_{40} \times y_5 & z_{14} &= t_{29} \times y_2 & z_{15} &= t_{42} \times y_9 \\
z_{16} &= t_{45} \times y_{14} & z_{17} &= t_{41} \times y_8
\end{aligned} \tag{6.25}$$

In (6.25), t_{25} through t_{40} coincides to the part where the multiplicative inverse in $GF((2^2)^2)$ is calculated.

6.1.5. Canright

This `SubBytes` implementation finds the multiplicative inverse in $GF(((2^2)^2)^2)$. $GF(2^2)$ is defined using the basis $\{x_0^2, x_0\}$ with x_0 being a root of the irreducible polynomial $x^2 + x + 1$ over $GF(2)$. Then, $GF((2^2)^2)$ uses the basis $\{y_0^4, y_0\}$, in which y_0 is a root of the irreducible polynomial $x^2 + x + x_0^2$ over $GF(2^2)$. Lastly, to get $GF(((2^2)^2)^2)$ from $GF((2^2)^2)$, it uses the basis $\{z_0^{16}, z_0\}$, where z_0 is a root of the irreducible polynomial $x^2 + x + x_0y_0$ over $GF((2^2)^2)$.

The matrices that map an element of $GF(2^8)$ to $GF(((2^2)^2)^2)$ (T) and vice versa (T^{-1}) are presented in (6.26).

$$\begin{aligned}
T &= \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \\
T^{-1} &= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}
\end{aligned} \tag{6.26}$$

After the conversion, an element $D \in GF(((2^2)^2)^2)$ is represented as in (6.27).

$$\begin{aligned}
D &= d_1 z_0^{16} + d_0 z_0 \\
d_1 &= d_{11} y_0^4 + d_{10} y_0 & d_0 &= d_{01} y_0^4 + d_{00} y_0 \\
d_{11} &= d_{111} x_0^2 + d_{110} x_0 & d_{10} &= d_{101} x_0^2 + d_{100} x_0 \\
d_{01} &= d_{011} x_0^2 + d_{010} x_0 & d_{00} &= d_{001} x_0^2 + d_{000} x_0
\end{aligned}$$

$$D = \begin{bmatrix} d_{111} \\ d_{110} \\ d_{101} \\ d_{100} \\ d_{011} \\ d_{010} \\ d_{001} \\ d_{000} \end{bmatrix} \tag{6.27}$$

Since the implementation uses normal bases to find the multiplicative inverse at every level, the rest follows from (6.8), (6.9) and (6.10), in which the irreducible polynomials are adopted as the ones specified for this implementation.

There are a couple of additional functions to be explicitly defined, which are derived from multiplication algorithms in (6.8), (6.9) and (6.10). They aid in simplifying the calculation of the multiplicative inverse. The first one is squaring and scaling by $x_0 y_0$ in $GF((2^2)^2)$ which is used in (6.8). For an element $E = e_1 y_0^4 + e_0 y_0$ in $GF((2^2)^2)$, squared and scaled output is defined as in (6.28).

$$(e_1 + e_0)^{-1}y_0^4 + (e_0 \times x_0^2)^{-1}y_0 \quad (6.28)$$

The second one is scaling by x_0^2 in $GF(2^2)$ to be used in (6.9) (inversion). Since squaring in $GF(2^2)$ is the same as multiplicative inversion, which is just switching the coefficients of x_0^2 and x_0 , it does not require additional logic. After all, scaling an element $F = f_1x_0^2 + f_0x_0$ by x_0^2 in $GF(2^2)$ is reduced to (6.29).

$$f_0x_0^2 + (f_1 + f_0)x_0 \quad (6.29)$$

The last one is multiplication followed by scaling by x_0^2 found in (6.9) (multiplication). Applying this operation to elements $G = g_1x_0^2 + g_0x_0$ and $H = h_1x_0^2 + h_0x_0$ in $GF(2^2)$ results in (6.30).

$$\begin{aligned} &((g_1 + g_0)(h_1 + h_0) + \alpha)x_0^2 + (g_1h_1 + \alpha)x_0 \\ &\alpha = g_0h_0 \end{aligned} \quad (6.30)$$

6.1.6. Nogami

Y. Nogami et al. [45] point out that in finite fields, multiplication is more easily implemented using normal basis, although inversion is more easily implemented using polynomial basis. As a starting point, Y. Nogami et al. [45] use the irreducible polynomial $x^2 + x + 1$ over $GF(2)$, $x^2 + x + x_0$ over $GF(2^2)$ and $x^2 + x + x_0^2y_0$ over $GF((2^2)^2)$, in which x_0 is a root of $x^2 + x + 1$, y_0 is a root of $x^2 + x + x_0$ and z_0 is a root of $x^2 + x + x_0^2y_0$. Eventually, $GF(2^2)$ is defined over $GF(2)$ with the basis $\{x_0^2, x_0\}$, $GF((2^2)^2)$ is defined over $GF(2^2)$ with the basis $\{y_0, 1\}$, and $GF(((2^2)^2)^2)$ is defined over $GF((2^2)^2)$ with the basis $\{z_0^{16}, z_0\}$.

In order to reduce the number of ones contained in the matrices of (6.12), Y. Nogami et al. [45] rearrange the multiplicative inversion in $GF(((2^2)^2)^2)$ of (6.8) for normal basis to give an output in polynomial basis. To demonstrate, (6.8) is rewritten for an element $D = d_1z_0^{16} + d_0z_0$ in $GF(((2^2)^2)^2)$ of *Nogami* as (6.31).

$$\begin{aligned} D^{-1} &= \alpha^{-1}(d_0z_0^{16} + d_1z_0) \\ \alpha &= d_1d_0 + (d_1 + d_0)^2x_0^2y_0 \end{aligned} \quad (6.31)$$

And since $z_0^{16} = z_0 + 1$, (6.31) becomes (6.32).

$$D^{-1} = \alpha^{-1}((d_1 + d_0)z_0 + d_0) \quad (6.32)$$

Hence the output of (6.31) is converted into polynomial basis in (6.32) with the addition of an extra XOR gate. It is seen as a worthy trade in favor of reducing the complexity of conversion matrix multiplications. This changing of basis during an operation is called the Mixed Bases (MB) approach by Y. Nogami et al. [45]. Subsequently, the conversion matrices are provided in (6.33).

$$T = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (6.33)$$

$$T^{-1} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Y. Nogami et al. [45] then apply this approach of changing the basis at the output of a function to $GF((2^2)^2)$ multiplication and multiplicative inversion. The need to do this stems from the simplicity of multiplication in polynomial basis and multiplicative inversion in normal basis compared to other bases. But, looking at (6.32) one could see multiplications followed by inversion and vice versa. For that reason, if one wishes to implement a multiplication in polynomial basis that is followed by an inversion in normal basis and the other way around, basis changes similar to the one in (6.32) should be employed.

The inversion in (6.32) is followed by two parallel multiplications. So, taking (6.9) and making the same adjustments to it that are done to (6.31) results in the following equations in (6.34).

$$\begin{aligned}
E &= e_1 y_0^4 + e_0 y_0 \quad E \in GF((2^2)^2) \\
E^{-1} &= \delta^{-1}(e_0 y_0^4 + e_1 y_0) = \delta^{-1}((e_0 + e_1) y_0 + e_0) \\
\delta &= e_1 e_0 + (e_1 + e_0)^2 x_0
\end{aligned} \tag{6.34}$$

Note that, the identity $y_0^4 = y_0 + 1$ is used to derive the polynomial basis form of E^{-1} (6.34). The redundancy of an additional XOR gate introduced in (6.34) is outweighed by the simpler structure of finding the multiplicative inverse in normal basis.

There is a multiplication $d_1 d_0$ and a scaling by $x_0^2 y_0$ in α of (6.32) before the inversion. If those multiplications are done in polynomial basis, they should be converted to normal basis before being operated by (6.34). These modifications to multiplication in (6.5) are shown below in (6.35) and (6.36).

$$\begin{aligned}
F &= f_1 y_0 + f_0 \quad G = g_1 y_0 + g_0 \quad F, G \in GF((2^2)^2) \\
F \times G &= (f_1 g_0 + f_0 g_1 + f_1 g_1) y_0 + f_0 g_0 + f_1 g_1 x_0 \\
&= (f_0 g_0 + f_1 g_1 x_0) y_0^4 \\
&\quad + ((f_0 + f_1)(g_0 + g_1) + f_1 g_1 x_0) y_0
\end{aligned} \tag{6.35}$$

$$\begin{aligned}
F \times x_0^2 y_0 &= f_1 x_0^2 x_0 y_0^4 + ((f_0 + f_1) x_0^2 + f_1 x_0^2 x_0) y_0 \\
&= f_1 y_0^4 + (f_0 x_0^2 + f_1 x_0) y_0
\end{aligned} \tag{6.36}$$

Notice that, the identities $y_0^4 = y_0 + 1$ and $x_0^3 = 1$ are made use of in deriving (6.35) and (6.36). A trade off similar to the one in (6.34) exists in (6.35) and (6.36) as well.

Finally, squaring in $GF((2^2)^2)$ polynomial basis, squaring in $GF(2^2)$ normal basis, scaling by x_0 in $GF(2^2)$ normal basis and scaling by x_0^2 in $GF(2^2)$ normal basis can be reduced further from their respective multiplication algorithms ((6.5) and (6.10)). Scaling by x_0^2 in $GF(2^2)$ normal basis and squaring in $GF(2^2)$ normal basis is exactly the same as *Canright*. The remaining two are given by (6.37) and (6.38).

$$\begin{aligned}
P &= p_1 y_0 + p_0 \quad P \in GF((2^2)^2) \\
P^2 &= p_1^2 y_0 + p_0^2 + p_1^2 x_0
\end{aligned} \tag{6.37}$$

$$\begin{aligned}
Q &= q_1 x_0^2 + q_0 x_0 \quad Q \in GF(2^2) \\
&= (q_1 + q_0) x_0^2 + q_1 x_0
\end{aligned} \tag{6.38}$$

6.1.7. Nekado

This implementation of `SubBytes` finds the multiplicative inverse in $GF((2^4)^2)$ similar to *Wolkerstorfer*. The difference in the paper of K. Nekado et al. [46] arises from its innovative approach while representing $GF(2^4)$. It makes this approach based on an interesting fact about the roots of the polynomial $x^4 + x^3 + x^2 + x + 1$ which is irreducible over $GF(2)$. If x_0 is assumed to be one of its roots, then the following derivation (6.40) about x_0^5 can be made from (6.39).

$$x_0^4 + x_0^3 + x_0^2 + x_0 + 1 = 0 \quad (6.39)$$

$$(x_0 + 1)(x_0^4 + x_0^3 + x_0^2 + x_0 + 1) = x_0^5 + 1 = 0 \quad (6.40)$$

$$x_0^5 = 1$$

Which means order of the multiplicative subgroup formed by the powers of x_0 is five. What is interesting about this is, those five elements include every root of $x^4 + x^3 + x^2 + x + 1$ and $x_0^0 = 1$, namely, $\{x_0^{2^3}, x_0^{2^2}, x_0^2, x_0, 1\}$. But since it also includes five powers of x_0 , $\{x_0^{2^3}, x_0^{2^2}, x_0^2, x_0, 1\} = \{x_0^4, x_0^3, x_0^2, x_0, 1\}$. This is also true because, $x_0^{2^2} = x_0^4$ and $x_0^{2^3} = x_0^8 = x_0^5 x_0^3 = x_0^3$. Furthermore, as two and five are coprime, roots of $x^4 + x^3 + x^2 + x + 1$, $\{x_0^{2^3}, x_0^{2^2}, x_0^2, x_0\}$ are all generators of the same group, $\{x_0^{2^3}, x_0^{2^2}, x_0^2, x_0, 1\}$. As a conclusion of all these factors, the normal basis and the four polynomial bases that can be composed from the roots of $x^4 + x^3 + x^2 + x + 1$ and 1 are the $\binom{5}{4}$ combinations of the five elements, $\{x_0^4, x_0^3, x_0^2, x_0, 1\}$. This is explained by the equations (6.41), (6.42), (6.43), (6.44) and (6.45) below.

Normal basis

$$\{x_0, x_0^2, x_0^{2^2} = x_0^4, x_0^{2^3} = x_0^8 = x_0^3\} = \{x_0^4, x_0^3, x_0^2, x_0\} \quad (6.41)$$

First polynomial basis

$$\{x_0^3, x_0^2, x_0, 1\} \quad (6.42)$$

Second polynomial basis

$$\{(x_0^2)^3 = x_0^6 = x_0, (x_0^2)^2 = x_0^4, x_0^2, 1\} = \{x_0^4, x_0^2, x_0, 1\} \quad (6.43)$$

Third polynomial basis

$$\{(x_0^{2^2})^3 = x_0^{12} = x_0^2, (x_0^{2^2})^2 = x_0^8 = x_0^3, x_0^{2^2} = x_0^4, 1\} = \{x_0^4, x_0^3, x_0^2, 1\} \quad (6.44)$$

Fourth polynomial basis

$$\begin{aligned} \{(x_0^{2^3})^3 = x_0^{24} = x_0^4, (x_0^{2^3})^2 = x_0^{16} = x_0, x_0^{2^3} = x_0^8 = x_0^3, 1\} \\ = \{x_0^4, x_0^3, x_0, 1\} \end{aligned} \quad (6.45)$$

K. Nekado et al. [46], make a case for redundantly using all of $\{x_0^4, x_0^3, x_0^2, x_0, 1\}$ to represent the elements in $GF(2^4)$ rather than constraining oneself with any of the five bases shown above. They call this representation (or basis) the Redundantly Represented Basis (RRB). Rationale behind this redundancy is explained by the ensuing arguments:

- Elements of $GF(2^4)$ are not uniquely defined in RRB. For example, $x_0^4 + x_0^2$ is the same as $x_0^3 + x_0 + 1$ because of (6.39). However, this becomes advantageous when every element can be represented by five-bit numbers with Hamming weights being equal to two or less.
- Any of the five standard bases can be easily converted to and from the RRB. Converting from any of the other basis to RRB is done via representing the missing bit by zero. To explain it further, let $D \in GF(2^4)$ be represented in normal basis as $d_3x_0^8 + d_2x_0^4 + d_1x_0^2 + d_0x_0 = d_2x_0^4 + d_3x_0^3 + d_1x_0^2 + d_0x_0$ where $d_2, d_3, d_1, d_0 \in GF(2)$. This element can be turned into a binary number with four bits as $(d_2 \ d_3 \ d_1 \ d_0)_2$. D is equal to $d_2x_0^4 + d_3x_0^3 + d_1x_0^2 + d_0x_0$ in RRB as well. But when it comes to representing it in binary, one should also include the coefficient of 1 which is equal to zero. As a result, it becomes $(d_2 \ d_3 \ d_1 \ d_0 \ 0)_2$. As for converting from RRB to one of the other bases, it is simply accomplished through getting rid of the extra bit by distributing it over other bits. To give an example, let $D \in GF(2^4)$ be represented by RRB as $d_4x_0^4 + d_3x_0^3 + d_2x_0^2 + d_1x_0 + d_0$. It can be converted to say the first polynomial basis

$\{x_0^3, x_0^2, x_0, 1\}$ as $(d_3 + d_4)x_0^3 + (d_2 + d_4)x_0^2 + (d_1 + d_4)x_0 + (d_0 + d_4)$ using the equality $x_0^4 = x_0^3 + x_0^2 + x_0 + 1$ in accordance with (6.39).

- This flexibility of RRB provides freedom in choosing the matrices T and T^{-1} that convert the AES $GF(2^8)$ to $GF((2^4)^2)$ and vice versa. Thereupon, these matrices can be optimized to contain as many zeros as possible to make the conversions more efficient when it comes to the number of logic gates necessary.
- Another advantage of RRB is the ease of squaring and multiplication by constants x_0^4, x_0^3, x_0^2, x_0 and 1. Both of these operations can theoretically be performed with no delay, because they just correspond to rearranging of wires. To demonstrate, let $D \in GF(2^4)$ be represented by RRB as $d_4x_0^4 + d_3x_0^3 + d_2x_0^2 + d_1x_0 + d_0$, then $D^2 = (d_4x_0^4 + d_3x_0^3 + d_2x_0^2 + d_1x_0 + d_0)^2 = d_4^2(x_0^4)^2 + d_3^2(x_0^3)^2 + d_2^2(x_0^2)^2 + d_1^2x_0^2 + d_0^2 = d_4x_0^8 + d_3x_0^6 + d_2x_0^4 + d_1x_0^2 + d_0 = d_2x_0^4 + d_4x_0^3 + d_1x_0^2 + d_3x_0 + d_0$. Similarly, if D is multiplied by a constant x_0^2 , it would be $D \times x_0^2 = (d_4x_0^4 + d_3x_0^3 + d_2x_0^2 + d_1x_0 + d_0) \times x_0^2 = d_4x_0^6 + d_3x_0^5 + d_2x_0^4 + d_1x_0^3 + d_0x_0^2 = d_2x_0^4 + d_1x_0^3 + d_0x_0^2 + d_4x_0 + d_3$.
- Lastly, the RRB algorithms for multiplication and inversion in $GF(2^4)$, equations of which are given in (6.46), (6.47) and (6.48), are comparable to other bases in terms of critical path delay.

$$\begin{aligned}
 D &= d_4x_0^4 + d_3x_0^3 + d_2x_0^2 + d_1x_0 + d_0 \\
 E &= e_4x_0^4 + e_3x_0^3 + e_2x_0^2 + e_1x_0 + e_0 \\
 F &= f_4x_0^4 + f_3x_0^3 + f_2x_0^2 + f_1x_0 + f_0 \\
 D, E, F &\in GF(2^4)
 \end{aligned} \tag{6.46}$$

$$\begin{aligned}
D \times E &= (d_3e_1 + d_2e_2 + d_0e_4 + d_1e_3 + d_4e_0)x_0^4 \\
&\quad + (d_2e_1 + d_1e_2 + d_4e_4 + d_0e_3 + d_3e_0)x_0^3 \\
&\quad + (d_1e_1 + d_0e_2 + d_3e_4 + d_4e_3 + d_2e_0)x_0^2 \\
&\quad + (d_0e_1 + d_4e_2 + d_2e_4 + d_3e_3 + d_1e_0)x_0 \\
&\quad + (d_4e_1 + d_3e_2 + d_1e_4 + d_2e_3 + d_0e_0) \\
&= ((d_1 + d_3)(e_1 + e_3) + (d_0 + d_4)(e_0 + e_4))x_0^4 \\
&\quad + ((d_1 + d_2)(e_1 + e_2) + (d_0 + d_3)(e_0 + e_3))x_0^3 \\
&\quad + ((d_0 + d_2)(e_0 + e_2) + (d_3 + d_4)(e_3 + e_4))x_0^2 \\
&\quad + ((d_0 + d_1)(e_0 + e_1) + (d_2 + d_4)(e_2 + e_4))x_0 \\
&\quad + ((d_1 + d_4)(e_1 + e_4) + (d_2 + d_3)(e_2 + e_3))
\end{aligned} \tag{6.47}$$

$$\begin{aligned}
F^{-1} &= (F \times F^4)^{-1}F^4 \\
&= ((f_4x_0^4 + f_3x_0^3 + f_2x_0^2 + f_1x_0 + f_0)(f_1x_0^4 + f_2x_0^3 \\
&\quad + f_3x_0^2 + f_4x_0 + f_0))^2(f_1x_0^4 + f_2x_0^3 + f_3x_0^2 + f_4x_0 + f_0) \\
&= (f_0 + f_1 + (f_0 + f_4)(f_0 + f_3)(f_2 + f_3))x_0^4 \\
&\quad + (f_0 + f_2 + (f_0 + f_3)(f_0 + f_1)(f_1 + f_4))x_0^3 \\
&\quad + (f_0 + f_3 + (f_0 + f_2)(f_0 + f_4)(f_1 + f_4))x_0^2 \\
&\quad + (f_0 + f_4 + (f_0 + f_1)(f_0 + f_2)(f_2 + f_3))x_0 \\
&\quad + ((f_0 + f_1)(f_0 + f_4)\overline{(f_2 + f_3)} \\
&\quad + (f_0 + f_2)(f_0 + f_3)\overline{(f_1 + f_4)})
\end{aligned} \tag{6.48}$$

Through the transformation $F^{-1} = (F \times F^4)^{-1}F^4$ in (6.48), the multiplicative inversion in $GF(2^4)$ is turned into a multiplicative inversion in $GF(2^2)$ and a multiplication in $GF(2^4)$. The reason $F^5 \in GF(2^2)$ is because $2^4 - 1 = (2^2 - 1)(2^2 + 1)$ and $(2^2 + 1) = 5$ is a prime. Moreover, according to the definitions made in the Finite Fields chapter, $\forall F \in \{GF(2^4) \setminus GF(2^2)\} - \{0\}$, F generates a multiplicative group of order m that is a multiple of 5 defined in the region $2^4 - 1 = 15 \geq m \geq 5 = (2^2 + 1)$ and F^5 generates a multiplicative group of order $k = \frac{m}{\gcd(m,5)}$. In conclusion, since $\gcd(m,5) = 5$, any multiplicative group, F^5 generates must be between $(2^2 - 1) = 3 \geq k \geq 1 = \frac{5}{\gcd(5,5)}$. In fact, this method can be used to reduce the inversion in $GF(2^{2n})$ to an inversion in $GF(2^n)$

by rewriting F^{-1} , $\forall F \in GF(2^n)$ as $(F \times F^{2^n})^{-1} F^{2^n}$. After all is said and done, the multiplicative inversion in $GF(2^2)$ is equivalent to a squaring, while the multiplication in $GF(2^4)$ is already given in (6.47).

In addition to the adoption of RRB for the reasons above, K. Nekado et al. [46] make use of the Mixed Bases in the paper of Y. Nogami et al. [45] and after calculating the multiplicative inverse in normal basis, it converts the output to polynomial basis. Employment of RRB and MB this way opens up many options for the conversion matrices.

An extensive search is made in this project to find possible T and $A \times T^{-1}$ candidates that have the least number of ones in any row and in total. In doing so, every representation mentioned so far has been considered for both the input and the output of the multiplicative inversion. It was finally decided on to use the matrices in (6.49).

$$T = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad (6.49)$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

T converts an element in $GF(2^8)$ to an element $Q = (q_{13}x_0^4 + q_{12}x_0^3 + q_{11}x_0 + q_{10})y_0^{16} + (q_{03}x_0^4 + q_{02}x_0^2 + q_{01}x_0 + q_{00})y_0 \in GF((2^4)^2)$. $(q_{13}x_0^4 + q_{12}x_0^3 + q_{11}x_0 + q_{10})$ and $(q_{03}x_0^4 + q_{02}x_0^2 + q_{01}x_0 + q_{00})$ are then converted to RRB as $(q_{13}x_0^4 + q_{12}x_0^3 + q_{14}x_0^2 + q_{11}x_0 + q_{10})$ and $(q_{03}x_0^4 + q_{04}x_0^3 + q_{02}x_0^2 + q_{01}x_0 + q_{00})$ in which $q_{14} = q_{04} = 0$. After the multiplicative inverse of Q is calculated as $Q^{-1} = (q'_{14}x_0^4 + q'_{13}x_0^3 + q'_{12}x_0^2 + q'_{11}x_0 + q'_{10})y_0 + (q'_{04}x_0^4 + q'_{03}x_0^3 + q'_{02}x_0^2 + q'_{01}x_0 + q'_{00}) \in GF((2^4)^2)$ it is converted from RRB form to $((q'_{14} + q'_{12})x_0^4 + (q'_{13} + q'_{12})x_0^3 + (q'_{11} + q'_{12})x_0 + q'_{10} + q'_{12})y_0 + ((q'_{04} + q'_{03})x_0^4 + (q'_{02} + q'_{03})x_0^2 + (q'_{01} + q'_{03})x_0 + q'_{00} + q'_{03}) \cdot y_0$ appearing in the preceding

equations is chosen to be a root of the irreducible polynomial $y^2 + y + x_0^4$ over $GF(2^4)$. Ultimately, this representation was converted back into $GF(2^8)$ with T^{-1} to carry out the rest of the `SubBytes` operation.



7. DESIGN FLOW

After laying out the finite arithmetic models of five implementations, the main goal is to simulate virtual power analysis attacks on circuits based on these implementations using different logic gate topologies.

As a first step towards this goal, SABL and ADDT topologies are applied to a few of the most frequently used Boolean functions, creating custom logic gates that are then compared against standard cell libraries.

7.1. DESIGN OF CUSTOM GATES

Starting point for designing both SABL and ADDT was the XH018 D_CELLS digital standard cell library of X-FAB. The devices belonging to this library have 0.18-micron drawn gate length and 1.8-Volt DC supply voltage rating. They are described as standard speed and power among other libraries present in the XH018 technology.

Being the most frequently used ones, some timing, voltage, load and size characteristics of NAND (NA2X1), NOR (NO2X1) and XNOR (EN2X1) gates in D_CELLS library are listed in Table 7.1 and Table 7.2. These characteristics are used as a basis for the corresponding SABL and ADDT gates.

Table 7.1. Timing, voltage and load characteristics of D_CELLS

	T_{PLH}	T_{PHL}	V_{IFL}	V_{IFH}	C_{in}	C_{out}	I_{avg}
NA2X1	169 ps	133 ps	53 mV	73 mV	5.9 fF	8.32 fF	1.41 μ A
NO2X1	290 ps	84 ps	59 mV	63 mV	6.69 fF	7.38 fF	1.47 μ A
EN2X1	307 ps	228 ps	34 mV	46 mV	10.17 fF	8.07 fF	2.45 μ A

Table 7.2. Transistor sizes of D_CELLS

	W_{PDN}	L_{PDN}	W_{PUN}	L_{PUN}
NA2X1	$0.8 \mu m$	$180 nm$	$1 \mu m$	$180 nm$
NO2X1	$0.66 \mu m$	$180 nm$	$1.4 \mu m$	$180 nm$
EN2X1 (nand2)	$0.4 \mu m$	$180 nm$	$0.5 \mu m$	$180 nm$
EN2X1 (o2na2)	$0.78 \mu m$	$180 nm$	$1.44 \mu m$	$180 nm$

The columns of Table 7.1 and Table 7.2 can be described as such:

- T_{PLH} is propagation delay during a low to high transition.
- T_{PHL} is propagation delay during a high to low transition.
- V_{IFL} is input feedthrough during a low to high transition.
- V_{IFH} is input feedthrough during a high to low transition.
- C_{in} is total input capacitance.
- C_{out} is open circuit output capacitance.
- W_{PDN} is the channel width of the devices contained in the pull-down network.
- L_{PDN} is the channel length of the devices contained in the pull-down network.
- W_{PUN} is the channel width of the devices contained in the pull-up network.
- L_{PUN} is the channel length of the devices contained in the pull-up network.
- I_{avg} is the average current drawn from the supply.
- These parameters are calculated by averaging the results of every possible input transition.
- The transition threshold voltage for propagation delays is taken as $0.9 V$.
- The measurements are made with an output load of $25 fF$.
- EN2X1 is defined as $A \oplus B = \overline{(\overline{A \wedge B}) \wedge (A \vee B)}$. It consists of two logic gates that realize Boolean functions $\overline{(A \wedge B)}$ (nand2) and $\overline{C \wedge (A \vee B)}$ (o2na2). Its configured such that the output of nand2 is connected to C input of o2na2.

7.1.1. Schematic

Two gates are designed for each topology, a NAND and an XNOR gate. Six different Boolean functions (AND, NAND, OR, NOR, XOR and XNOR) can be obtained via permutating the inputs and outputs of these two gates. Switching the inputs of the NAND gate creates an OR gate, while switching the outputs of each gate results in its inversion.

The processes by which schematic device sizes are determined through are summarized below.

- The output inverters that are common in all four gates are taken from the NOT (INX1) gate of D_CELLS for compatibility with the standard cells.
- The PMOS transistors that are driven by the **CLK** signal in all four gates are equivalent in size to the PMOS of INX1, since the size of that PMOS is suitable for charging a node by itself. Moreover, symmetry between the nodes that they charge is another concern. So, they have to match in size with each other for a given gate.
- Every NMOS in the pull-down network of a gate must be identical with each other because of symmetry. Only exception to this could be the NMOS driven by the **CLK** signal. The effect of pull-down network on switching speed and input capacitance is given in Figure 7.1 and Figure 7.2.
- The sense-amplifiers of SABL gates can be formed by using minimum-size transistors. PMOS transistors of the sense-amplifiers are not used to charge nodes but to keep them at VDD voltage, so they can be designed to be small. Additionally, since the sense-amplifiers are the last link of a discharge chain, size of their NMOS transistors contributes the least in terms of the gate's switching speed. This is demonstrated for comparison against the pull-down network in Figure 7.3 and Figure 7.4. Lastly, small size of the sense-amplifiers somewhat increases the clock feedthrough as shown in Figure 7.5.

Following these guidelines, initial schematics for four gates were created. Consecutively, device sizes were readjusted during the design of the layout. This continued iteratively until target specifications are met with minimum possible size and within the layout limitations.

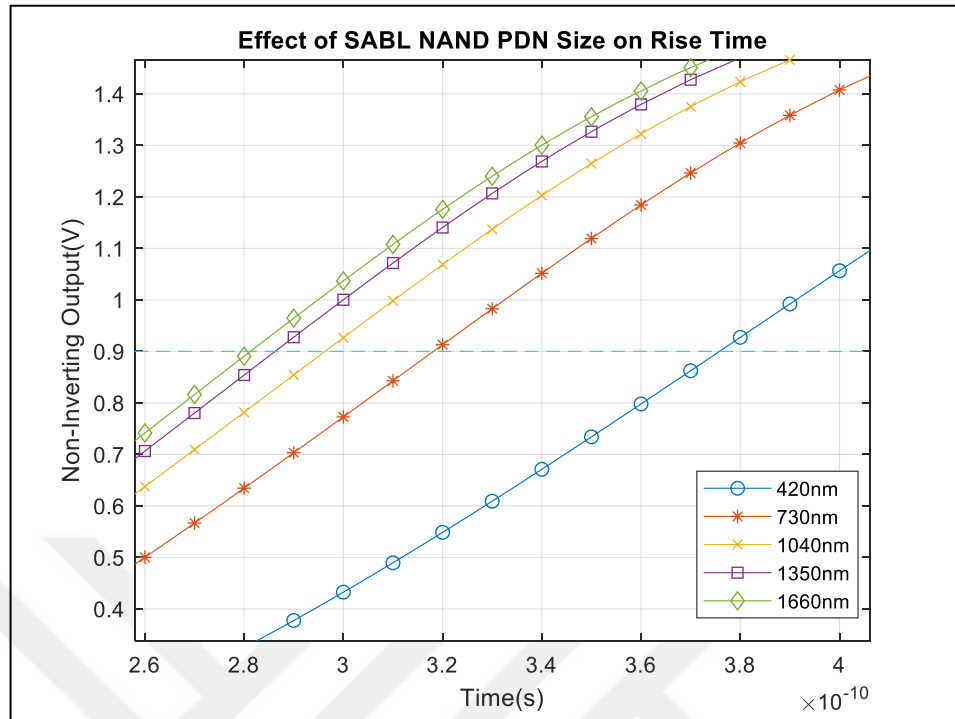


Figure 7.1. SABL NAND non-inverting output for PDN widths 420nm-1660nm

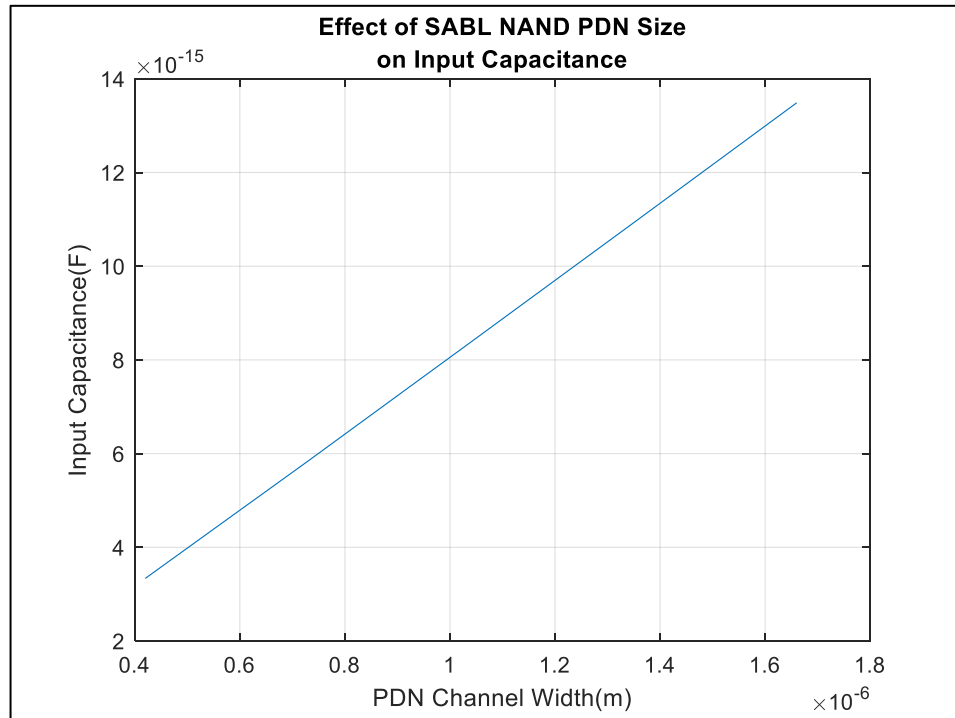


Figure 7.2. SABL NAND input capacitance for PDN widths 420nm-1660nm

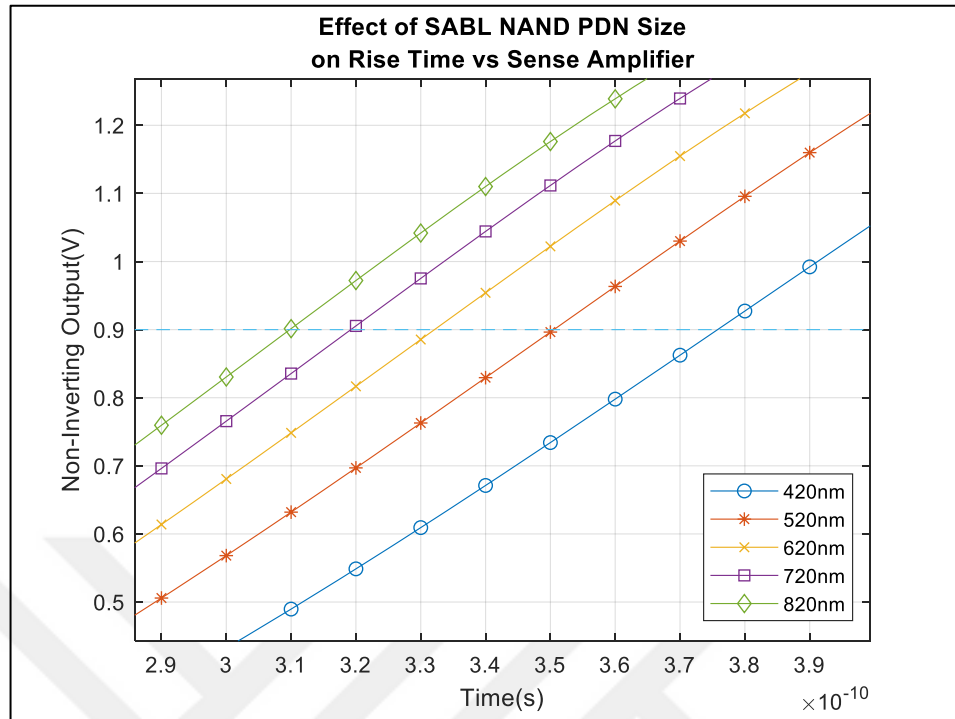


Figure 7.3. SABL NAND non-inverting output for PDN widths 420nm-820nm

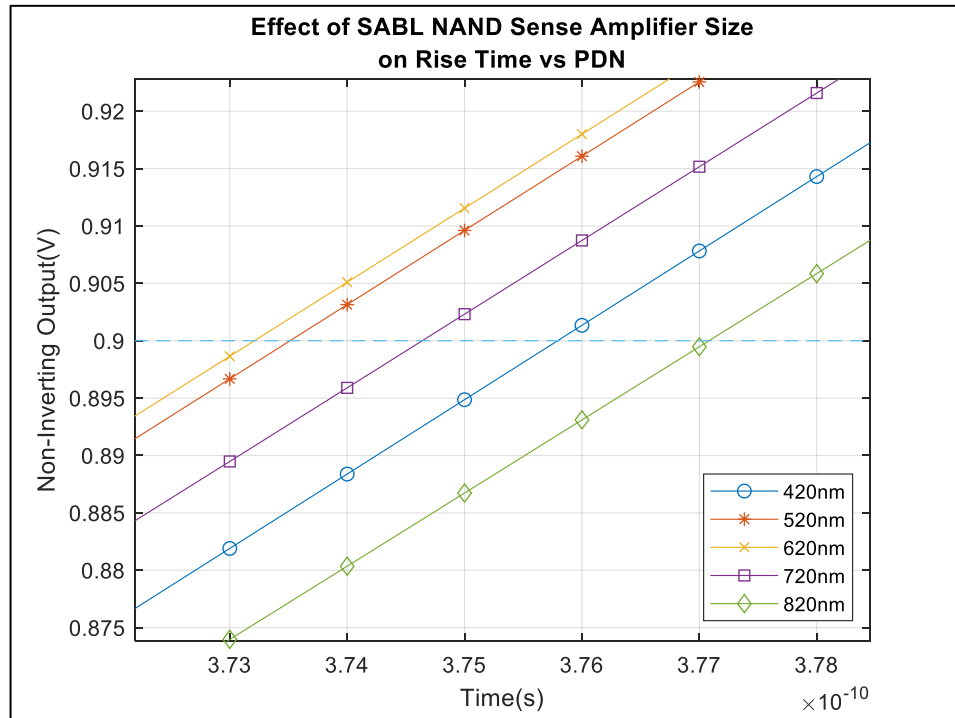


Figure 7.4. SABL NAND non-inverting output for sense-amp widths 420nm-820nm

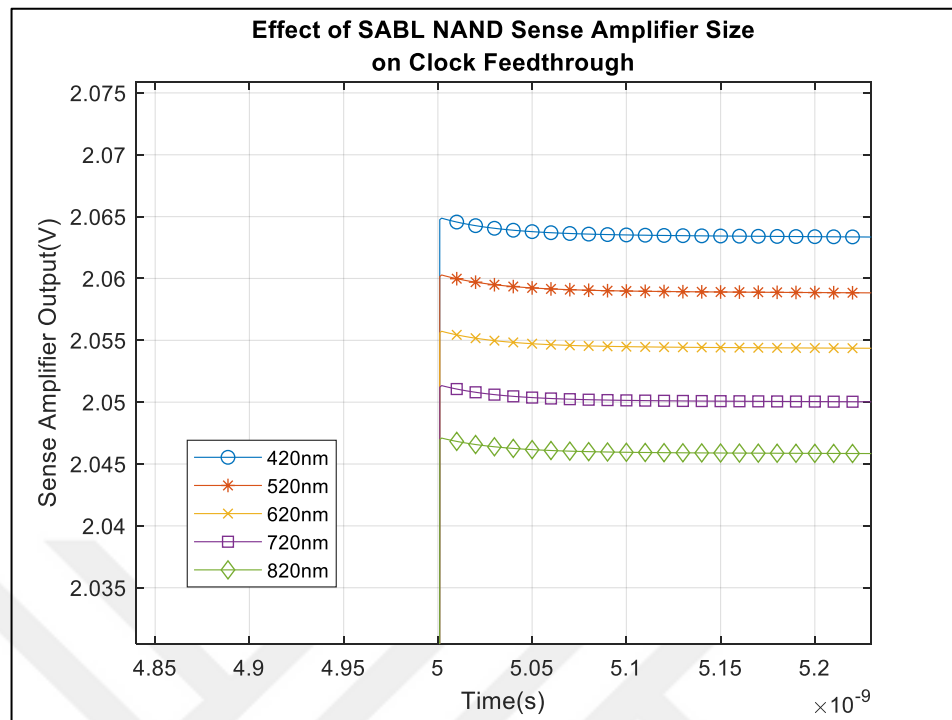


Figure 7.5. SABL NAND **n1** node for sense-amp widths 420nm-820nm

7.1.2. Layout

As a starting point, the height of the custom gates is chosen to be $4.88 \mu m$ in compliance with the standard library cells. Similarly, the heights of N+ Implant and P+ Implant layers are set as $1.86 \mu m$ and $2.4 \mu m$, which is the same as they are for the standard library cells. These heights correspond to a maximum N+ Active layer height of $1.08 \mu m$ and P+ Active layer height of $1.62 \mu m$. These in turn represent maximum channel widths if the Polysilicon layer is drawn as a rectangle that intersects the Active layer vertically, meaning transistors are lined up horizontally. This kind of transistor placement is the most meaningful one as the circuit design consists of rows with fixed height in which cells of same height and different widths are placed side by side and uninterrupted N+ and P+ implant layers run horizontally.

The second course of action was to look for Eulerian paths in the schematic. For SABL, there is an Eulerian path that starts from one output and ends at its complement, which goes through every one of six PMOS transistors in the gate, while there does not exist one for

ADDT. The longest Eulerian paths of PMOS transistors in ADDT gates contain two of them. When it comes to Eulerian paths that include NMOS transistors, the longest one contains eight transistors for SABL NAND and SABL XNOR, seven for ADDT XNOR and nine for ADDT NAND. But all these NMOS Eulerian paths include pull-down transistor of only one output inverter, which makes an uninterrupted N⁺ Active layer between two outputs impossible, unlike the SABL P⁺ Active layer.

It is apparent that there are longer Eulerian paths of NMOS transistors than of PMOS transistors. The same relation is true for the number of NMOS and PMOS transistors, so NMOS transistors determine the size of the layout for the most part. There are two main ways to design the layouts in this case. The first way is to place every PMOS at the top and every NMOS at the bottom as regular. But this puts an upper limit to the width of a PMOS channel at around $1.62\ \mu\text{m}$ and NMOS channel at around $1.08\ \mu\text{m}$. While $1.62\ \mu\text{m}$ PMOS channel width is enough, $1.08\ \mu\text{m}$ channel width for an NMOS might limit the overall speed of the gate as it is shown in Figure 7.1.

The other way is to reserve a section in the layout where its full height is utilized as N⁺ Implantation layer, therefore giving a lot of room for NMOS transistors placed in any orientation. This does not increase the total width of the layout compared to the first way of regular placement in the case of SABL NAND, SABL XNOR and ADDT XNOR. Because for these gates, the layout width is dominated by NMOS transistors and there is not one Eulerian path that covers them all, which results in even longer NMOS sequences. However, this way of placement would be greatly disadvantageous in terms of size for ADDT NAND. That is because of ADDT NAND's comparatively greater number of PMOS transistors and the presence of an Eulerian path that contains almost every NMOS transistor of it, which makes possible a regular placement, where PMOS and NMOS transistors placed tightly takes approximately equal amount of horizontal space. Using instead the second way of placement would leave a lot of empty space and produce a considerably bigger layout in comparison to other three custom gates. The downside of this, as it was mentioned, is a reduction in switching speed. Yet, the fact that there are more XNOR/XOR gates than NAND/AND/NOR/OR gates for all five SubBytes implementations as shown in Table 7.9 lessens the negative consequences of this design choice for ADDT NAND.

Lastly, it is decided that the pull-down transistors of the output inverters are to be left out of the Eulerian paths. Including only one of them in the path does not really decrease the width of the layout, not to mention it results in a small negative effect on symmetry.

Everything explained so far leads to the layouts and device sizes given in Table 7.3, Table 7.4, Table 7.5, Table 7.6, Figure 7.6, Figure 7.7, Figure 7.8 and Figure 7.9.



Table 7.3. SABL NAND Sizes

Pull-Down Network of Figure 4.7		M1	M2	M3	M4	M13	M14
	W	1.66 μm	1.66 μm	1.66 μm	1.66 μm	1.66 μm	1.66 μm
	L	185 nm	185 nm	185 nm	185 nm	185 nm	185 nm
Charge and Discharge Switches of Figure 4.7			M5	M6	M7	M12	
	W		1.68 μm	1.68 μm	1.66 μm	220 nm	
	L		185 nm	185 nm	185 nm	1 μm	
Sense-Amplifier of Figure 4.7			M8	M9	M10	M11	
	W		420 nm	420 nm	420 nm	420 nm	
	L		180 nm	180 nm	180 nm	180 nm	
Inverters of Figure 4.7		Inverter Pull-Up Transistor			Inverter Pull-Down Transistor		
	W	1.68 μm			660 nm		
	L	185 nm			180 nm		

Table 7.4. SABL XNOR Sizes

Pull-Down Network of Figure 4.8		M1	M2	M3	M4	M13	M14
	W	1.66 μm	1.66 μm	1.66 μm	1.66 μm	1.66 μm	1.66 μm
	L	185 nm	185 nm	185 nm	185 nm	185 nm	185 nm
Charge and Discharge Switches of Figure 4.8			M5	M6	M7	M12	
	W		1.68 μm	1.68 μm	1.6 μm	220 nm	
	L		185 nm	185 nm	180 nm	1 μm	
Sense-Amplifier of Figure 4.8			M8	M9	M10	M11	
	W		420 nm	420 nm	420 nm	420 nm	
	L		180 nm	180 nm	180 nm	180 nm	
Inverters of Figure 4.8		Inverter Pull-Up Transistor			Inverter Pull-Down Transistor		
	W	1.68 μm			660 nm		
	L	185 nm			180 nm		

Table 7.5. ADDT NAND Sizes

Pull-Down Network of Figure 4.9	<table border="1" data-bbox="727 450 1358 622"> <thead> <tr> <th></th> <th>M1</th> <th>M2</th> <th>M3</th> <th>M4</th> </tr> </thead> <tbody> <tr> <th>W</th> <td>1.14 μm</td> <td>1.14 μm</td> <td>1.14 μm</td> <td>1.14 μm</td> </tr> <tr> <th>L</th> <td>185 nm</td> <td>185 nm</td> <td>185 nm</td> <td>185 nm</td> </tr> </tbody> </table> <table border="1" data-bbox="727 678 1358 851"> <thead> <tr> <th></th> <th>M5</th> <th>M6</th> <th>M7</th> <th>M8</th> </tr> </thead> <tbody> <tr> <th>W</th> <td>1.14 μm</td> <td>1.14 μm</td> <td>1.14 μm</td> <td>1.14 μm</td> </tr> <tr> <th>L</th> <td>185 nm</td> <td>185 nm</td> <td>185 nm</td> <td>185 nm</td> </tr> </tbody> </table>		M1	M2	M3	M4	W	1.14 μm	1.14 μm	1.14 μm	1.14 μm	L	185 nm	185 nm	185 nm	185 nm		M5	M6	M7	M8	W	1.14 μm	1.14 μm	1.14 μm	1.14 μm	L	185 nm	185 nm	185 nm	185 nm
	M1	M2	M3	M4																											
W	1.14 μm	1.14 μm	1.14 μm	1.14 μm																											
L	185 nm	185 nm	185 nm	185 nm																											
	M5	M6	M7	M8																											
W	1.14 μm	1.14 μm	1.14 μm	1.14 μm																											
L	185 nm	185 nm	185 nm	185 nm																											
Charge and Discharge Switches of Figure 4.9	<table border="1" data-bbox="727 983 1358 1155"> <thead> <tr> <th></th> <th>M9</th> <th>M10</th> <th>M11</th> <th>M12</th> </tr> </thead> <tbody> <tr> <th>W</th> <td>1.49 μm</td> <td>1.49 μm</td> <td>1.49 μm</td> <td>1.49 μm</td> </tr> <tr> <th>L</th> <td>185 nm</td> <td>185 nm</td> <td>185 nm</td> <td>185 nm</td> </tr> </tbody> </table> <table border="1" data-bbox="727 1211 1358 1384"> <thead> <tr> <th></th> <th>M13</th> <th>M14</th> <th>M15</th> <th>M16</th> </tr> </thead> <tbody> <tr> <th>W</th> <td>1.49 μm</td> <td>1.49 μm</td> <td>1.49 μm</td> <td>1.49 μm</td> </tr> <tr> <th>L</th> <td>185 nm</td> <td>185 nm</td> <td>185 nm</td> <td>185 nm</td> </tr> </tbody> </table>		M9	M10	M11	M12	W	1.49 μm	1.49 μm	1.49 μm	1.49 μm	L	185 nm	185 nm	185 nm	185 nm		M13	M14	M15	M16	W	1.49 μm	1.49 μm	1.49 μm	1.49 μm	L	185 nm	185 nm	185 nm	185 nm
	M9	M10	M11	M12																											
W	1.49 μm	1.49 μm	1.49 μm	1.49 μm																											
L	185 nm	185 nm	185 nm	185 nm																											
	M13	M14	M15	M16																											
W	1.49 μm	1.49 μm	1.49 μm	1.49 μm																											
L	185 nm	185 nm	185 nm	185 nm																											
Inverters of Figure 4.9	<table border="1" data-bbox="655 1599 1428 1827"> <thead> <tr> <th></th> <th>Inverter Pull-Up Transistor</th> <th>Inverter Pull-Down Transistor</th> </tr> </thead> <tbody> <tr> <th>W</th> <td>1.68 μm</td> <td>660 nm</td> </tr> <tr> <th>L</th> <td>185 nm</td> <td>180 nm</td> </tr> </tbody> </table>		Inverter Pull-Up Transistor	Inverter Pull-Down Transistor	W	1.68 μm	660 nm	L	185 nm	180 nm																					
	Inverter Pull-Up Transistor	Inverter Pull-Down Transistor																													
W	1.68 μm	660 nm																													
L	185 nm	180 nm																													

Table 7.6. ADDT XNOR Sizes

Pull-Down Network of Figure 4.10		M1	M2	M3	M4	M5	M6
	W	1.66 μm	1.66 μm	1.66 μm	1.66 μm	1.66 μm	1.66 μm
	L	185 nm	185 nm	185 nm	185 nm	185 nm	185 nm
Charge and Discharge Switches of Figure 4.10		M7	M8	M9	M10	M11	M12
	W	1.68 μm	1.68 μm	1.68 μm	1.68 μm	1.68 μm	1.68 μm
	L	185 nm	185 nm	185 nm	185 nm	185 nm	185 nm
Inverters of Figure 4.10		Inverter Pull-Up Transistor		Inverter Pull-Down Transistor			
	W	1.68 μm		660 nm			
	L	185 nm		180 nm			

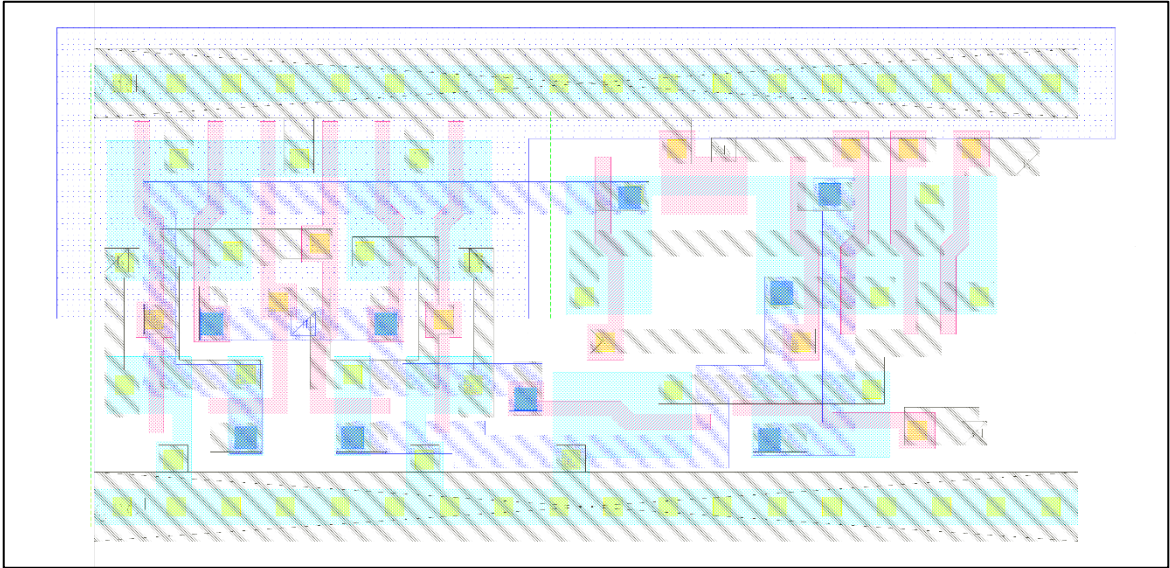


Figure 7.6. SABL NAND

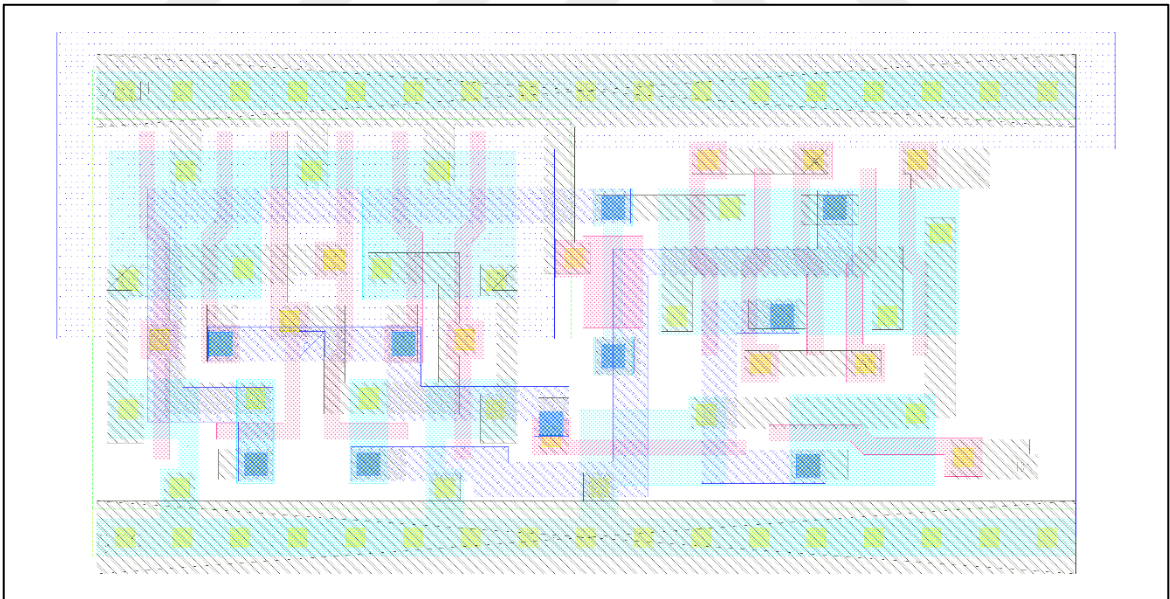


Figure 7.7. SABL XNOR

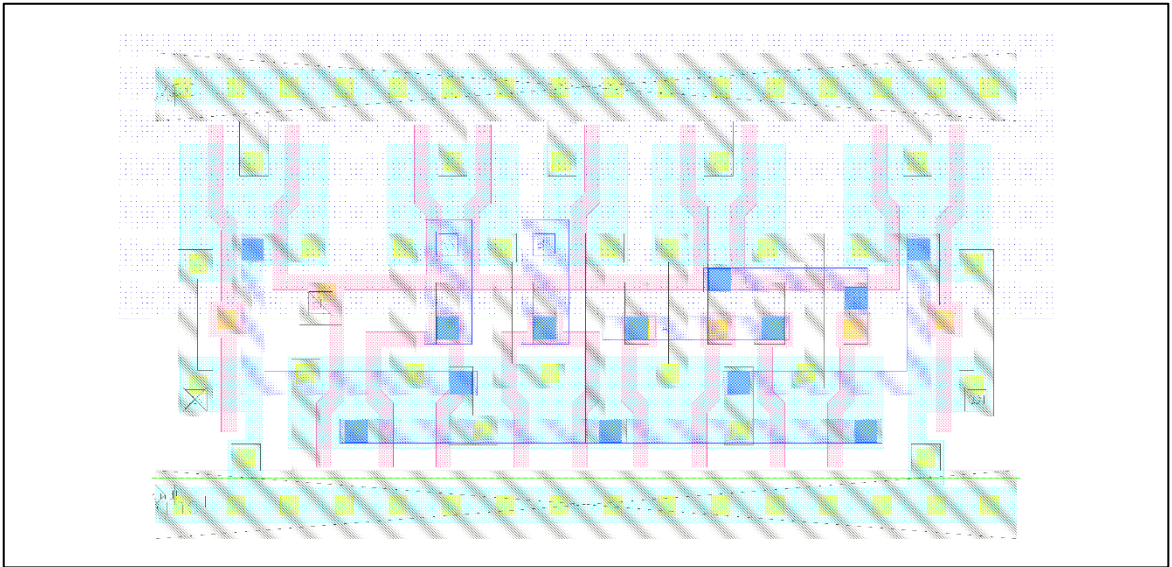


Figure 7.8. ADDT NAND

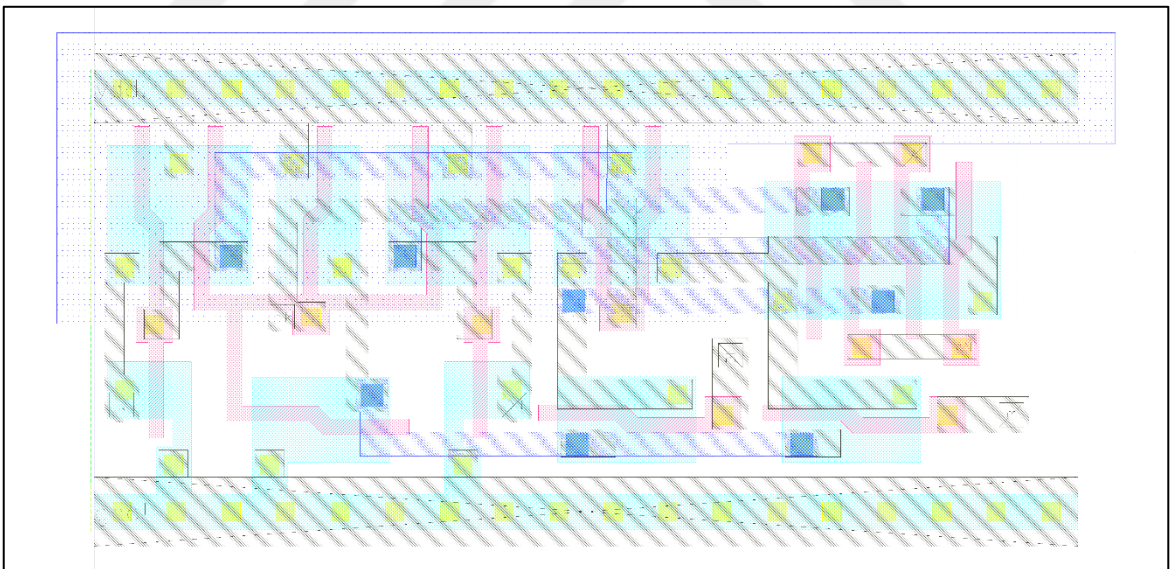


Figure 7.9. ADDT XNOR

A few remarks about the layouts:

- Minimum channel width of the Sense-amplifier transistors is limited by the minimum contact size, so they are 420 nm wide (*minimum contact dimension* +

minimum contact to active distance $\times 2 = 220 \text{ nm} + 100 \text{ nm} \times 2 = 420 \text{ nm}$). Increasing the distance between contacts to allow minimum channel width of 220 nm instead, would require a larger area in this case.

- The transistors with 185 nm channel lengths are the ones with non-rectangular gate areas. The gates are drawn this way to squeeze more transistors horizontally. The extra channel length caused by this irregular shape is negligible, but the increased channel width somewhat makes difference. Nevertheless, increasing the size of these transistors mainly serves to pass the Layout Versus Schematic verifications.

For the sake of comparison, several parameters that belong to the newly created gates are measured and summed up below in Table 7.7.

Table 7.7. Timing, voltage and load characteristics of SABL and ADDT

	T_{PE}	T_{PP}	V_{SFE}	V_{SFP}	C_{in}	C_{out}	I_{avg}
SABL NAND	237 ps	293 ps	19 mV	30 mV	13.49 fF	11.2 fF	5.91 μ A
SABL XNOR	278 ps	229 ps	20 mV	31 mV	13.45 fF	11.23 fF	5.91 μ A
ADDT NAND	256 ps	178 ps	28 mV	38 mV	11.94 fF	11.12 fF	6.09 μ A
ADDT XNOR	234 ps	185 ps	29 mV	37 mV	12.83 fF	11.12 fF	5.97 μ A

Timing and voltage characteristics in Table 7.7 have been chosen as to better represent the dynamic differential topology and still relate to static CMOS gates as much as possible. These differing characteristics are explained below.

- T_{PE} is average low-to-high propagation delay during evaluation phase.
- T_{PP} is average high-to-low propagation delay during precharge phase.
- V_{SFE} is the feedthrough of sense-amplifiers to the output of the inverters during evaluation phase.
- V_{SFP} is the feedthrough of sense-amplifiers to the output of the inverters during precharge phase.

7.2. DIGITAL DESIGN

During the digital design phase, four different semi-custom design methods using four different set of logic gates are adopted to build eleven circuits in total from five implementations (*Wolkerstorfer*, *Boyar*, *Canright*, *Nogami* and *Nekado*). These four methods are listed below.

- In the **first design** all five implementations are built using a restricted set of D_CELLS standard cell library. The set only contains standard power, standard speed and two-input NAND, AND, NOR, OR, XNOR, XOR and NOT (single input) gates to conform with the previously built custom SABL and ADDT gates. These five circuits are examined in contrast to each other, taking into consideration in order of importance: their power analysis resistance, size and power consumption attributes. Based on this examination, two (*Canright* and *Nekado*) out of five implementations are picked for three further designs.
- The **second design** utilizes the same library but does not restraint what type of logic gates are used during the synthesis step. Motivation behind this is to observe the impact of reductions that are possible through full usage of a standard library on the effectiveness of power analysis attacks. Because this would be the typical case when it comes to designing ASICs based on standard cell libraries.
- The last two designs (**third and fourth designs**) make use of the same netlist as the first one, since the custom SABL and ADDT gates are based on the gates used in the first design. On that grounds, the synthesis step is skipped. The third design is made with SABL gates and the fourth one is made with ADDT gates.

The summary of the circuits created during the digital design by pairing five implementations and four designs is given in Table 7.8. Later in the text, the naming convention in Table 7.8 will be used instead of explicitly specifying the design and the implementation.

Table 7.8. Circuits created with each design and implementation

	<i>Wolkerstorfer</i>	<i>Boyar</i>	<i>Canright</i>	<i>Nogami</i>	<i>Nekado</i>
First Design (Subset of D_CELLS)	<i>Wolkerstorfer_1</i>	<i>Boyar_1</i>	<i>Canright_1</i>	<i>Nogami_1</i>	<i>Nekado_1</i>
Second Design (D_CELLS)	-	-	<i>Canright_2</i>	-	<i>Nekado_2</i>
Third Design (SABL)	-	-	<i>Canright_3</i>	-	<i>Nekado_3</i>
Fourth Design (ADDT)	-	-	<i>Canright_4</i>	-	<i>Nekado_4</i>

7.2.1. Algorithmic and Behavioral Description

As a first step, five preceding implementations are developed algorithmically in MATLAB and verified for correct mapping of 8-bit inputs to 8-bit outputs as in the AES `SubBytes` operation.

After previous implementations are proven to operate properly, their behavioral descriptions are written in Verilog and tested in Xilinx Vivado Design Suite software.

7.2.2. Logic Synthesis

Netlists are synthesized in Cadence Genus Synthesis Solution software using Verilog behavioral descriptions of the previous step. Two different netlists are synthesized for four designs. One of the netlists is used by design number one, three and four, while the other is used in design number two.

Each circuit is designed to perform one `SubBytes` operation during a clock cycle. Clock period for the first and the second designs is determined according to this criterion with a

comfortable margin. The third and the fourth designs are assumed to work properly with the same clock period, since decision of the prior designs' clock period already takes into account the disparities between custom logic gates and the standard library cells. Not to mention, custom logic gates of the last two designs are created with the aim of having similar timing characteristics with the cells in the first designs, even though their typical load capacitances will vary. On that account, the synthesis results of the first design are taken as they are for the third and fourth designs with only difference being name of the logic gates in the netlists.

A few properties for the synthesis tool to work, concerning the input drivers, output loads and clock imperfections are set to inconsequential values. These are given as such:

- 20 MHz clock frequency
- 300 picoseconds of rising and falling edge clock slew.
- 500 picoseconds of delay between the rising edge of the clock and the arrival of a settled input from an external block.
- 500 picoseconds of delay between the settling of the output and the rising edge of the next clock cycle, which won't matter as the clock period is determined so that the output settles way earlier.
- The input is set to be driven by a standard speed, standard power and two-input, inverting multiplexer cell called MU2IX1. If the designed `SubBytes` is to be used in a cryptographic circuit that has both encryption and decryption capabilities, multiplicative inversion is usually shared between these operations. In those cases, the input to `SubBytes` is usually a multiplexer. In other cases, it may be a D flip-flop.
- Each output is set to drive an external load of 25 femtofarads. Through simulation and available timing library data, input loads of standard power, standard speed cells of the `D_CELLS` library are found to be around 5 femtofarads. This led to the somewhat arbitrary choice of 25-femtofarad output load for synthesis.

The second design employs two iterations of synthesis. Both synthesis iterations use more or less the same settings. Iteration number two is performed after the design goes through one place and route iteration. It uses the created layout as a reference and makes additional

post-place-and-route optimizations to the netlist. The other designs only go through synthesis once.

Besides creating layouts, the Verilog netlist files are also used in Cadence Virtuoso Schematic Editor to create schematics.

The logic cell and timing statistics of the first design is given below in Table 7.12. The same gate numbers apply to the third and fourth designs with the exception of NOT gates, since NOT gates are not needed in differential designs. The second design contains plethora of cells and their list is omitted for the sake of brevity. Still, to give some information, total gate counts of *Canright_2* and *Nekado_2* are 112 and 130, while their estimated delays are 20113 *ps* and 14396 *ps*.

Table 7.9. Gate statistics for the first, third and fourth designs

	<i>Boyar</i>	<i>Canright</i>	<i>Nekado</i>	<i>Nogami</i>	<i>Wolkerstorfer</i>
XNOR	78	64	56	63	61
XOR	30	34	50	42	41
NAND	22	13	34	25	37
AND	5	9	8	6	22
NOR	11	15	6	14	10
OR	6	7	7	4	3
NOT	7	5	9	13	7
Total	159	147	170	167	181
Estimated Delay	10768 <i>ps</i>	13682 <i>ps</i>	10251 <i>ps</i>	12676 <i>ps</i>	12067 <i>ps</i>

7.2.3. Place and Route

Place and route stage follows three different paths, one for the first design, one for the second design and one for the rest. Just as in the synthesis step, the second design goes through two iterations, whereas it is performed a single time for the others. These tasks are accomplished with the help of Cadence Innovus Implementation System software.

With that in mind, place and route steps of different designs are described in the succeeding subsections.

7.2.3.1. First Design

- First course of action is floorplanning. Since the netlist and the physical properties of the cells in the netlist are known, how much total area the circuit will take up is practically known as well. What is left to decide is the core aspect ratio, core utilization ratio and core to I/O boundary distance. Core is where the cells are placed. Thus, its aspect ratio and utilization ratio are bound by its routability. These properties are kept at software's defaults which are around 1 for aspect ratio and 0.7 for utilization ratio. Core to I/O boundary is set to 15 micrometers in order to fit the power and ground rings in between and reserve some space for input and output pins.
- Next thing in line is power routing. This includes placing power rings around the core and stripes over it so as to create a mesh on top of the circuit that distributes power evenly in the interest of reducing IR drops and electromigration. The rings are made of metal-6 layer on the sides and metal-5 layer on the top and bottom. The widths of the rings' metal lines are set to 2 micrometers. The distance between the GND and VDD rings are set to 1 micrometer. The distance of rings to the core is set to 2 micrometers. The vertical stripes are made of 4 equally-spaced 2-micrometer-wide metal-6 lines and the horizontal stripes are made of 4 equally-spaced 2-micrometer-wide metal-5 lines. VDD and GND stripes run in parallel 1 micrometer apart from each other. The last part of power routing is connecting cells to the power grid. This is also done automatically by the software tool through creating horizontal metal-1 lines that are set apart by the constant height of one cell inside the core area. Later on, cells are going to be placed between these lines.
- After power routing, the cells are placed in agreement with the netlist. Which means the cells that are connected are attempted to be placed in close proximity. But since cells cannot be too packed together as this would cause congestion issues during routing, the spaces are filled by filler cells to maintain continuity of n-well and implant layers.

- Following the placement, cells are routed. Routing is limited between metal-1 and metal-4 layers. The top two layers are reserved for power routing.
- Lastly, floating metal layers are placed everywhere around the circuit to attain a certain amount of metal density and uniformity. This is called metal filling.
- After the design is completed, a series of verifications are performed. These include DRC, connectivity, antenna violations and metal density verifications.

7.2.3.2. *Second Design*

The first iteration of place and route for the second design follows the same path with a few minor differences. These are listed below.

- The core aspect ratio is between 0.8 and 0.9 as default.
- It suffices to have two power stripes instead of four, since the design is smaller.
- Filler cells are not placed, since the placement of logic cells will be modified in the ensuing synthesis step.
- Since the second design will go through another synthesis and place and route after this one, metal filling step is skipped. It takes place after place and route of a design is finalized.
- During verification, metal density check is skipped and the design is exported in DEF format to be used in the second synthesis iteration

Fewer steps are performed in the second iteration. Because the design is imported from the second synthesis iteration as already placed. This placement occurs during the first place and route iteration and is further modified in the second synthesis iteration. In conjunction with these, the floorplanning, power routing and placement steps are mostly skipped. The differences between this iteration and preceding iteration of place and route are listed below.

- The only thing done in terms of power routing and placement is to place the filler cells according to the modified logic cells, and re-route the power connection of cells.
- As this is the final placement of cells, the interconnections between them are conclusively routed.
- Then, metal fill is added to complete the layout.

- Every verification in the first iteration is performed again with the addition of metal density verification.

7.2.3.3. Third and Fourth Designs

The main difference between last two design flows and the first one is routing. As it was made clear in the dynamic differential gates section, balancing capacitances of the complementary outputs is a big factor in power analysis resistance. Also mentioned is the fact that the output capacitances are dominated by metal lines connecting outputs to other cells. Therefore, complementary outputs should be connected to metal lines with equal areas.

The most straightforward way of accomplishing this is to run the metal lines belonging to complementary nets in parallel. Any other method would be too complicated to be accomplished by automated tools. Yet, creating the metal lines manually would be too difficult as well for a large netlist. For these reasons, the first mentioned method of running metal lines in parallel is adopted in the last two designs.

Using the first design flow as a reference again, a list of differences between the last two design flows and the first are given below.

- The core utilization ratio is set to 0.3, since higher ratios are tested and found to be unroutable by the routing tool as a result of congestion.
- Before placement, complementary wires are defined as differential pairs. Primary reason for this is to make Cadence Innovus skip routing those nets. Cadence Innovus' routing tool lacks space-aware routing capabilities. Hence, the routing of nets except those that belong to input, output and clock are going to be carried out in Cadence Virtuoso Space-Based Router (VSR), which is a part of Virtuoso Layout Suite.
- After the cells are placed and the aforementioned nets are routed, a DRC verification is performed as a last step in Cadence Innovus. The designs are then exported to Cadence Virtuoso.
- In Cadence Virtuoso Layout Suite, complementary nets are designated as differential pairs that are to be routed in parallel with the minimum distance allowed by foundry constraints. Then routing is proceeded in VSR as explained before.

- Routing in VSR is performed a couple of times. The tool fails to route every net without DRC and connectivity issues in one try. The first time, it is performed on all nets. Then the remaining faulty nets are rerouted until the violations are eliminated. Sometimes, when the tool is not able to route a few remaining nets, they are manually routed without adhering to differential pair rules if necessary.

7.2.3.4. Finalizing the Layout

After the layouts are designed, they are exported to Cadence Virtuoso, where they and the schematics are going to be brought together for a post-layout simulation. Here, one last DRC check and a Layout-vs-Schematic check is performed. Afterwards, parasitic extraction of resistances and capacitances for each layout is executed through Cadence Quantus Extraction Solution. Parasitic resistances are extracted using a mix of square counting and resistance mesh technique where more accurate extraction of resistances between contacts is necessary. During square counting, maximum fracture length is not restricted.

When the extractions are completed, new layouts are created, adding the extracted parasitic components to the original layouts. These final layouts form the basis for succeeding post-layout simulations.

7.2.4. Post-Layout Simulation

Series connection of `AddRoundKey` trailed by `SubBytes` has two inputs (key and plaintext) and one output (ciphertext), all of which are 8-bit numbers. Even though `AddRoundKey` consists of only eight XOR gates which carry out a bit-wise XOR operation between key and plaintext, it is still included in the simulations. However, it should be noted this increases the number of input combinations from 2^8 to 2^{16} , which is very significant. All things considered, the final layouts for each design is simulated for every input combination (65536 times).

The simulations were 60 ns long, which included a 10 ns initial period during which clock signal was low. This is succeeded by a low-to-high transition of clock which initiates the evaluation phase and a high-to-low transition 25 ns later which initiates the precharge phase

that lasts 25 ns. All in all, the supply current is measured 65536 times during a single evaluation and precharge cycle. In order to save simulation time and space, parasitic elements that are negligible below 50 MHz are ignored and the results are retrieved starting from 10 ns mark with a 10 ps step size, which amounts to 5001 data points per supply current measurement.

7.2.4.1. Criteria for Assessment of Resistance

What determines the resistance of a cryptographic circuit against power analysis attacks as well as any other attack, is the number of tries necessary along with time it takes to perform a try to extract the key. For example, considering a brute-force attack on 128-bit variant of AES, it takes a maximum of 2^{128} and an average of 2^{127} encryptions to find the key, so however long it takes can be calculated by multiplying the number of encryptions with the average time it takes to perform an encryption.

On this basis, every implementation and every design in this project is compared with one another according to how many power measurements it takes to extract the key or otherwise how much the attack reduces the subset of keys in which the correct key exists. In doing so, various power-analysis techniques are experimented with.

Nonetheless, it should be recognized that, while power measurements from a real device is attempted to be mimicked in this project, they are far from accurately representing reality. The real goal in this attempt is to see how random/systematic noise and frequency-limiting affects the success of a power analysis attack. Eventually, the circuits are compared relative to each other.

7.2.4.2. Theoretical Estimations

The supply current measurements obtained from simulations are quite deterministic and precise. This means 65536 results the complete population. One can draw some conclusions from these results alone, without the need to imitate a real case scenario. In order to do this, the effect of noise on the correlation between hypothetical power consumption and real power consumption should be formulized. Let P_{tot} be total instantaneous power

consumption. According to what was established in Section 1.5, P_{tot} can be written as $P_{tot} = P_{exp} + P_{opno} + P_{elno} + P_{const} = P_{exp} + P_{noise} + P_{const}$. This leads to the following expression given in (7.1) for $\rho(H_j, M_j)$, the correlation between the j th column of H matrix which represents the hypothetical power consumption values for j th key and the j th column of M matrix which corresponds to the total power consumption at j th moment in (7.1).

$$\begin{aligned}
\rho(H_j, M_j) &= \rho(H_j, P_{tot,j}) = \rho(H_j, P_{exp,j} + P_{noise,j} + P_{const,j}) \\
&= \rho(H_j, P_{exp,j} + P_{noise,j}) = \frac{Cov(H_j, P_{exp,j} + P_{noise,j})}{\sqrt{Var(H_j)Var(P_{exp,j} + P_{noise,j})}} \\
&= \frac{E(H_j(P_{exp,j} + P_{noise,j})) - E(H_j)E(P_{exp,j} + P_{noise,j})}{\sqrt{Var(H_j)(Var(P_{exp,j}) + Var(P_{noise,j}))}} \\
&= \frac{E(H_j P_{exp,j}) + E(H_j P_{noise,j}) - E(H_j)E(P_{exp,j}) - E(H_j)E(P_{noise,j})}{\sqrt{Var(H_j)Var(P_{exp,j})} \sqrt{1 + \frac{Var(P_{noise,j})}{Var(P_{exp,j})}}} \quad (7.1) \\
&= \frac{E(H_j P_{exp,j}) - E(H_j)E(P_{exp,j})}{\sqrt{Var(H_j)Var(P_{exp,j})}} = \frac{\rho(H_j, P_{exp,j})}{\sqrt{1 + \frac{1}{SNR}}}
\end{aligned}$$

The simplifications in (7.1) can be attributed to the fact that $P_{const,j}$ does not affect the correlation and the assumption that $P_{noise,j}$ is independent from both H_j and $P_{exp,j}$ ($Cov(P_{exp,j}, P_{noise,j}) = 0$ and $Cov(H_j, P_{noise,j}) = 0$) which leads to the following equalities in (7.2) and (7.3).

$$\begin{aligned}
Var(P_{exp,j} + P_{noise,j}) &= Var(P_{exp,j}) + Var(P_{noise,j}) + 2Cov(P_{exp,j}, P_{noise,j}) \quad (7.2) \\
&= Var(P_{exp,j}) + Var(P_{noise,j})
\end{aligned}$$

$$E(H_j P_{exp,j}) = E(H_j)E(P_{noise,j}) + Cov(H_j, P_{noise,j}) = E(H_j)E(P_{noise,j}) \quad (7.3)$$

This assumption is fair, considering how $P_{noise,j}$ is defined in the beginning chapter.

The approximation $\rho(H_j, P_{tot,j})$ for the correct key and the spurious key with the highest $\rho(H_j, P_{exp,j})$ can then be used for ρ_0 and ρ_1 in (2.35) to generate an n as (7.4) which is an estimate for the number of measurements required to distinguish the correct key from the spurious keys with a probability $1 - \alpha$.

$$n = 3 + 8 \frac{(\Phi^{-1}(1 - \alpha))^2}{\left(\ln \frac{1 + \rho(H_{correct}, P_{tot,j})}{1 - \rho(H_{correct}, P_{tot,j})} - \ln \frac{1 + \rho(H_{spurious}, P_{tot,j})}{1 - \rho(H_{spurious}, P_{tot,j})} \right)^2} \quad (7.4)$$

After (7.4) is deducted, it is tested on *Canright_1* (*Canright* with a restricted subset of D_CELLS) for different hypothetical power consumption models with $\alpha = 10^{-5}$. The results of this analysis are presented in Table 7.10.

If $\rho(H_{spurious}, P_{tot,j})$ is greater than $\rho(H_{correct}, P_{tot,j})$, that model is completely ineffective as the attacker will always be wrong when the number of measurements are large enough. Because this invalidates the other models, the analysis is repeated this time for every implementation using only zero-value model. The outcomes are compiled in Table 7.11.

Table 7.10. Correlation coefficients and number of measurements estimate for *Canright_1*

for $\alpha = 10^{-5}$	$\rho(H_{correct}, P_{tot,j})$	$\rho(H_{spurious}, P_{tot,j})$	n^
Zero-value	0.0558	0.0407	120966
Hamming weight	0.0264	0.0467	N/A
Bit 0	0.0306	0.0445	N/A
Bit 1	0.0164	0.0384	N/A
Bit 2	0.0080	0.0116	N/A
Bit 3	0.0106	0.0152	N/A
Bit 4	0.0179	0.0462	N/A
Bit 5	0.0125	0.0434	N/A
Bit 6	0.0223	0.0440	N/A
Bit 7	0.0402	0.0580	N/A

Table 7.11. Correlation coefficients and number of measurements estimate for zero-value

for $\alpha = 10^{-5}$ and zero-value model	$\rho(H_{correct}, P_{tot,j})^$	$\rho(H_{spurious}, P_{tot,j})^*$	n^*
<i>Boyar_1</i>	0.0716	0.0422	31817
<i>Canright_1</i>	0.0558	0.0407	120953
<i>Nekado_1</i>	0.0734	0.0407	25648
<i>Nogami_1</i>	0.0684	0.0457	53241
<i>Wolkerstorfer_1</i>	0.0575	0.0490	382339
<i>Canright_2</i>	0.0537	0.0358	86445
<i>Canright_3</i>	0.0016	0.0039	N/A
<i>Canright_4</i>	0.0154	0.0095	807950
<i>Nekado_2</i>	0.0755	0.0501	42679
<i>Nekado_3</i>	0.0038	0.0066	N/A
<i>Nekado_4</i>	0.0042	0.0023	7117245

7.2.4.3. Preprocessing of the Results

As it was mentioned before, the simulations produce deterministic results. For this reason, they were processed to resemble power measurements from a real device before applying mock power analysis attacks.

It is said that, because of all the capacitive effects that are present in an integrated circuit, it is only possible to accurately observe the changes in power consumption which occur between different clock cycles and not during a single clock cycle. Besides parasitic effects, it makes sense that the voltage supplied to the circuit is intended to be stable and high frequency components are undesirable. This means the supply current measurements of a real circuit would be heavily filtered.

Additionally, as it is mentioned, the components in a circuit other than the one being attacked run in parallel and act as noise. While imitating the effects of these irrelevant switching activities, they are thought to have similar timing characteristics that are shared between many clocked digital circuits, which is having high activity for a short time a few moments after the clock edge and mostly settling down for the rest of the clock cycle. The way this kind of noise is implemented in this project was to take the average of the supply current for every input combination (all 65536 of them), fit a curve to these data points in order to acquire the supply current's general shape and use this shape as mean of a normally distributed noise. The curve is a Fourier series with eight terms, fitted using non-linear least squares regression method. Lastly, it is assumed arbitrarily that the attacked device uses the 128-bit variant of AES algorithm, so this random noise is added to the current consumption of 16 S-boxes that use the 16-byte key to be decrypted.

In consequence, the mock power measurements are designed to contain three periods of successive `AddRoundKey` and `SubBytes` operations. The key being attacked is used during the middle one of these periods. The rest are there as dummies to make the mixing effect that filtering has between adjacent clock periods more meaningful, hence are discarded to save memory.

7.2.4.4. Results of Correlation Coefficient-Based Attack

In order to perform a correlation coefficient-based attack, a random 16-byte key is designated and mock power measurements as a result of encrypting uniformly distributed random 16-byte plaintexts with this key are created as explained in the last chapter. The plaintexts form the P matrix while the middle portions of these mock power measurements form the M matrix of the correlation coefficient attack. The V matrix is the result of an `AddRoundKey` operation (XOR) on the key coupled with each plaintext followed by a `SubBytes` operation on the output of the `AddRoundKey` operation. The H matrix is calculated by assigning values to the elements of V matrix according to a hypothetical power consumption model. Finally, Pearson's correlation coefficient between each row of H and each column of P is calculated to get to the C matrix.

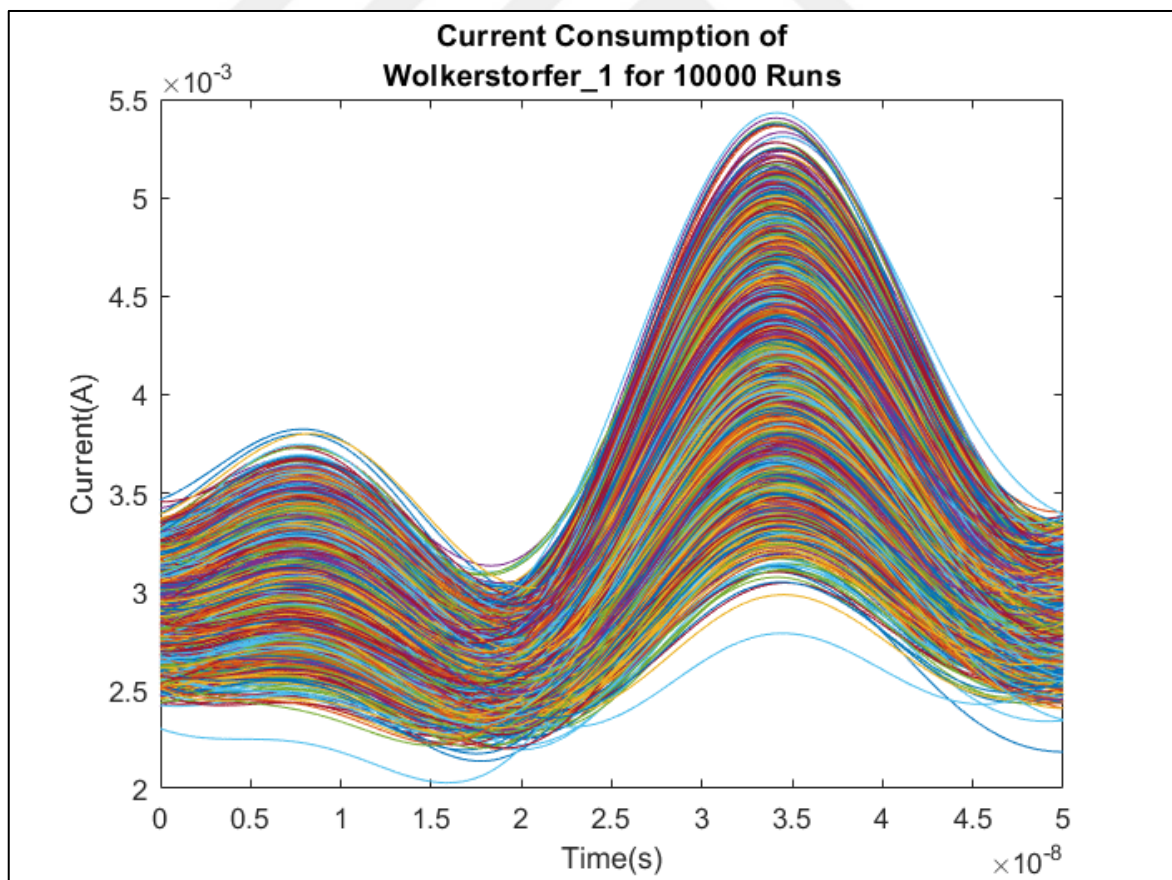


Figure 7.10. Current consumption of *Wolkerstorfer_1* for 10000 runs

As an example, 10000 power measurements are created for *Wolkerstorfer* implementation built according to the first design (*Wolkerstorfer_1*) with key in (7.5). These current consumption values are plotted altogether in Figure 7.10. The next step is to pick a byte of key to guess, as the attack is applied to one S-box at a time. The byte at column one and row one (22 in hexadecimal) is arbitrarily chosen for this purpose. If hypothetical power consumption matrix is based on zero-value model, this would result in the correlation coefficient matrix C , rows of which are plotted in Figure 7.11.

$$\begin{bmatrix} 22 & 39 & A1 & 5B \\ 3C & 43 & 7C & 98 \\ B5 & AA & 10 & 3E \\ 66 & C0 & BE & 42 \end{bmatrix} \quad (7.5)$$

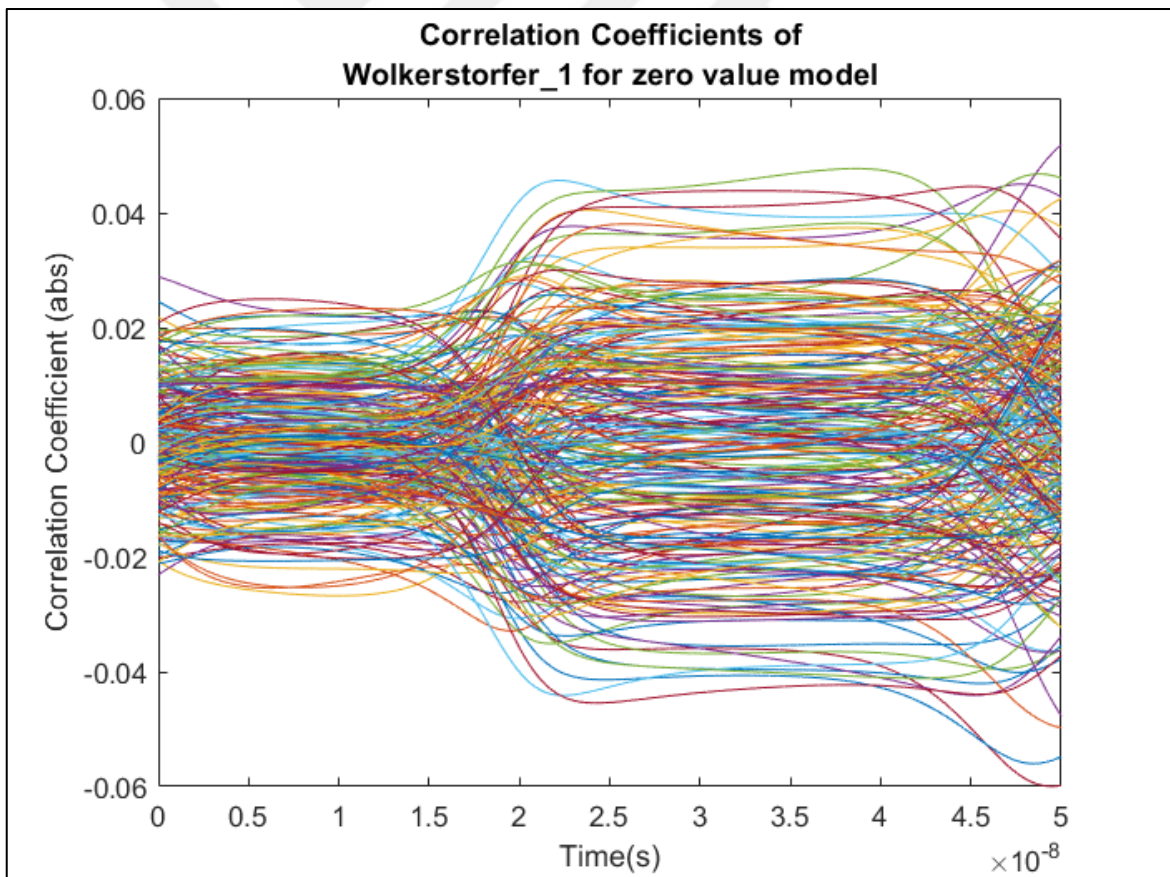


Figure 7.11. Correlation coefficients of *Wolkerstorfer_1* for zero-value model

There are 256 traces in Figure 7.11. Each of which belong to a key hypothesis from zero to 255, and they represent that key hypothesis' correlation with the power measurements at every instant. For the purpose of guessing the correct key, the maximum of each trace is calculated and plotted as in Figure 7.12, where the x-axis represents the key hypothesis and the y-axis represents the maximum absolute correlation coefficient for each trace along the duration of 50 ns. Guesses are made by sorting the keys with respect to their y-values in Figure 7.12 and going from top to bottom until the correct key is found. In this case, the hypothesis that generates the maximum correlation coefficient (34 in decimal, 22 in hexadecimal) matches the correct key. Thus, the correct key is guessed in one try.

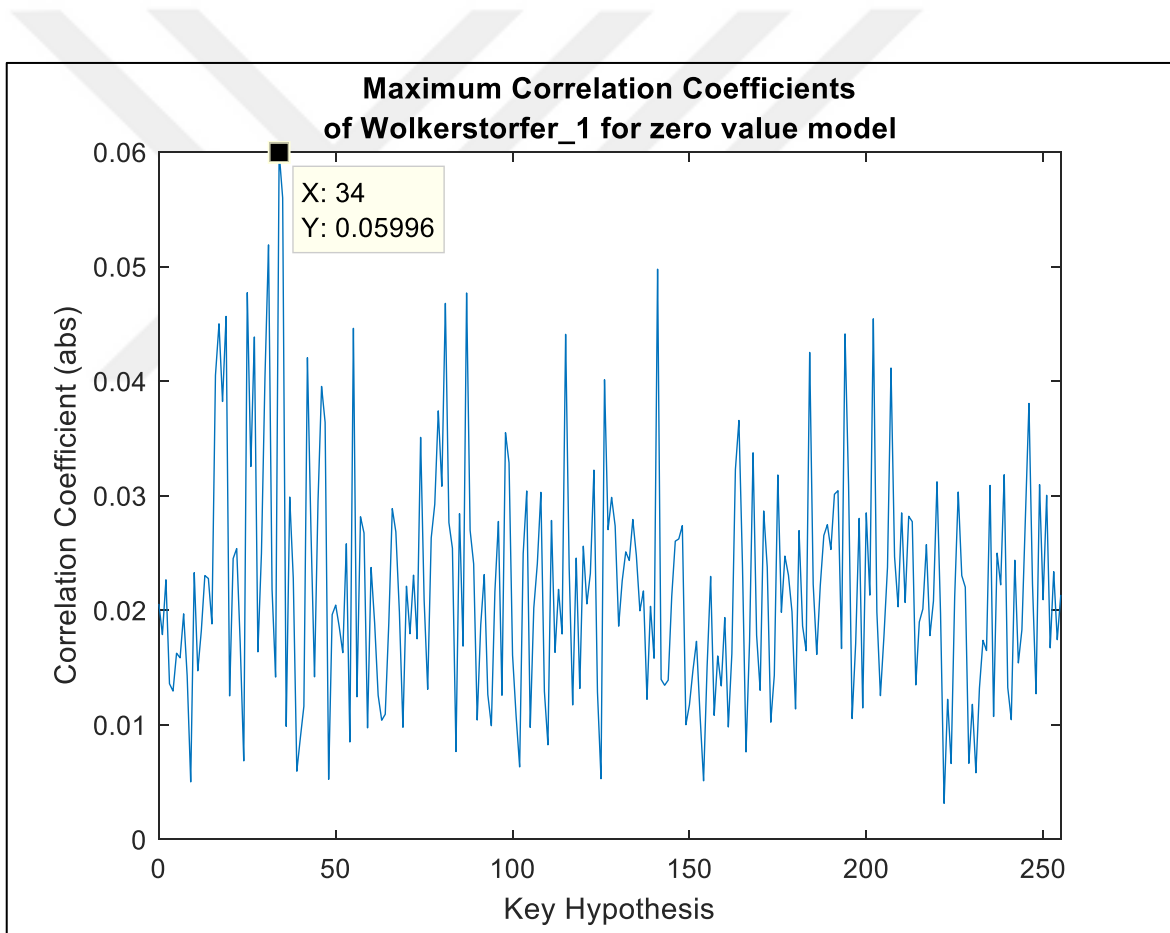


Figure 7.12. Maximum correlation coefficients of *Wolkerstorfer_1* for zero-value model

Since it is assumed that the internal knowledge of the circuits is unknown to the attacker, the hypothetical power consumption models available are chosen to be zero-value, Hamming weight and bit models. A comparison between these models for *Nekado_1* is given in Table

7.12. The way correlations and average guesses in Table 7.12 are calculated is by performing attacks on 32 different S-boxes using 10000 power measurements.

Table 7.12. Comparison of different hypothetical power consumption models

<i>*for 10000 measurements</i>	Average number of guesses until the correct key is found*	Average correlation coefficient for the correct key hypothesis*
Zero-value	1.47	0.0704
Hamming weight	163.75	0.0192
Bit 0	108.53	0.0234
Bit 1	142.88	0.0196
Bit 2	130.97	0.0213
Bit 3	160.53	0.0158
Bit 4	68.09	0.0298
Bit 5	120.16	0.0247
Bit 6	192.00	0.0133
Bit 7	70.94	0.0302

The table shows that zero-value model performs so much better than the other models. The only models other than zero-value model that show signs of significance are bit 4 and bit 7 models. The strength of zero-value model arises from the fact that during encryptions, the

circuits are initialized by setting plaintext, key and clock (if exists) to zero. Consequently, every encryption starts from the same initial state and in the presence of static CMOS gates, this initial state corresponds to encrypting plaintext 00 with key 00 and outputs a ciphertext of 63 in hexadecimal (99 in decimal). In this context, encrypting a plaintext with an equal key after that initial stage results in a very low power consumption. This is because `AddRoundKey` operation between equal plaintext-key pairs result in 00, as it is simply an XOR operation, and the output of `SubBytes` for the input value of 00 is 63 in hexadecimal.

The reason Hamming weight and bit models are so much worse as shown in Figure 7.12 is the S-box algorithm itself. Finite field arithmetic operations inside the S-box lead to an output that is mostly independent from the input and internal values. These models would work better if the transitions of the output bits were more dominant in overall power consumption, which means if the output capacitances were much greater than the input/output capacitors of the internal logic gates. This could be the case on a grander scale when a complete AES circuit is in question. In those cases, the output of an S-box could be connected to a large register or a data bus, which would affect the overall power consumption during output transitions. But this is not the case for the circuits in this project. Therefore, zero-value model will be the main focus while comparing power analysis resistance of these circuits.

After deciding on zero-value model, correlation coefficient attacks with different number of power measurements were performed on each circuit to test their resistance. The first comparison is done among five circuits built as stated in the first design using five different implementations (*Wolkerstorfer_1*, *Boyar_1*, *Canright_1*, *Nogami_1* and *Nekado_1*). The number of measurements was logarithmically swept between 10 and 100000. In addition to this, attacks were performed on 32 different S-boxes and the mean results were calculated as in the comparison of hypothetical power consumption models before. All in all, final outcomes are summarized in Figure 7.13 and Figure 7.14.

Following the examination of different implementations, *Nekado* and *Canright* were picked out of this group for the purpose of comparing different designs. The same analyses were carried out as before. The outcomes were given in Figure 7.15 and Figure 7.16 for *Canright*, and Figure 7.17 and Figure 7.18 for *Nekado*.

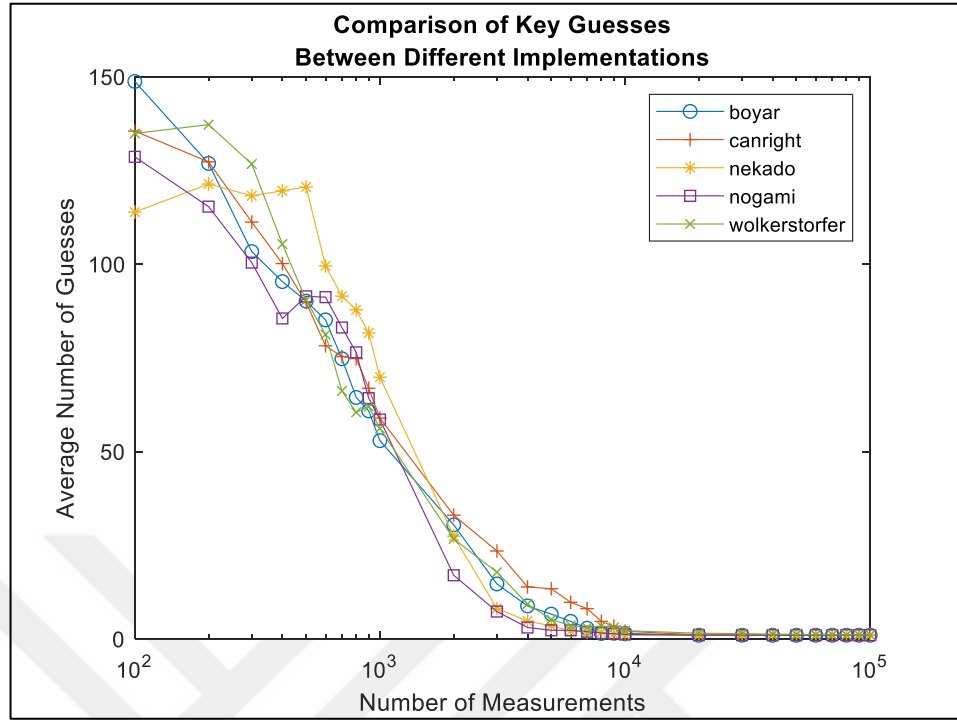


Figure 7.13. Comparison of key guesses between different implementations

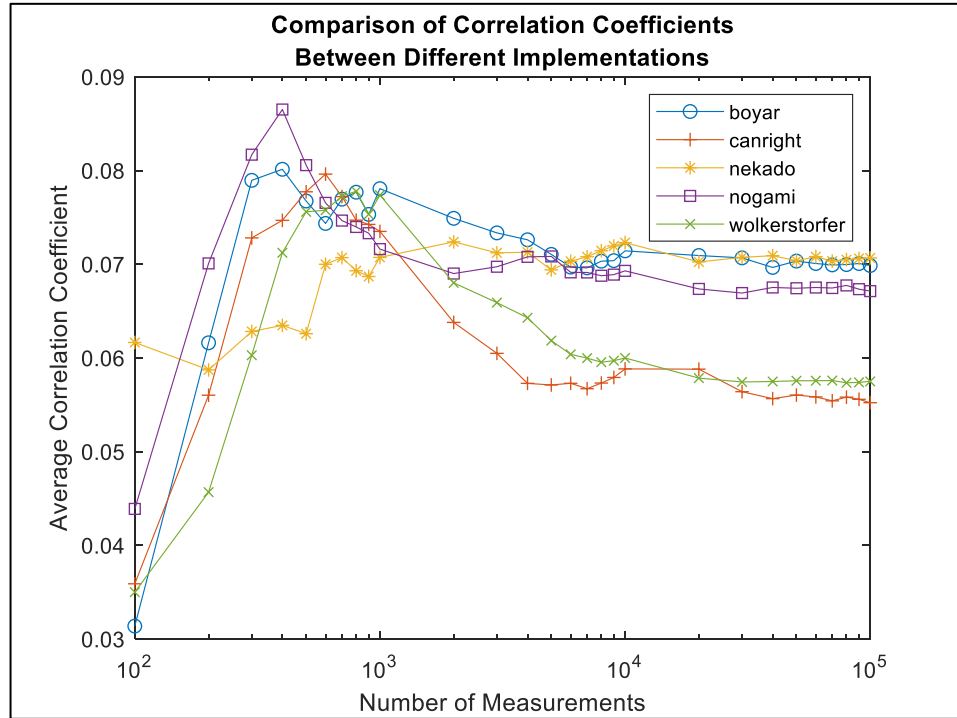


Figure 7.14. Comparison of correlation coefficients between different implementations

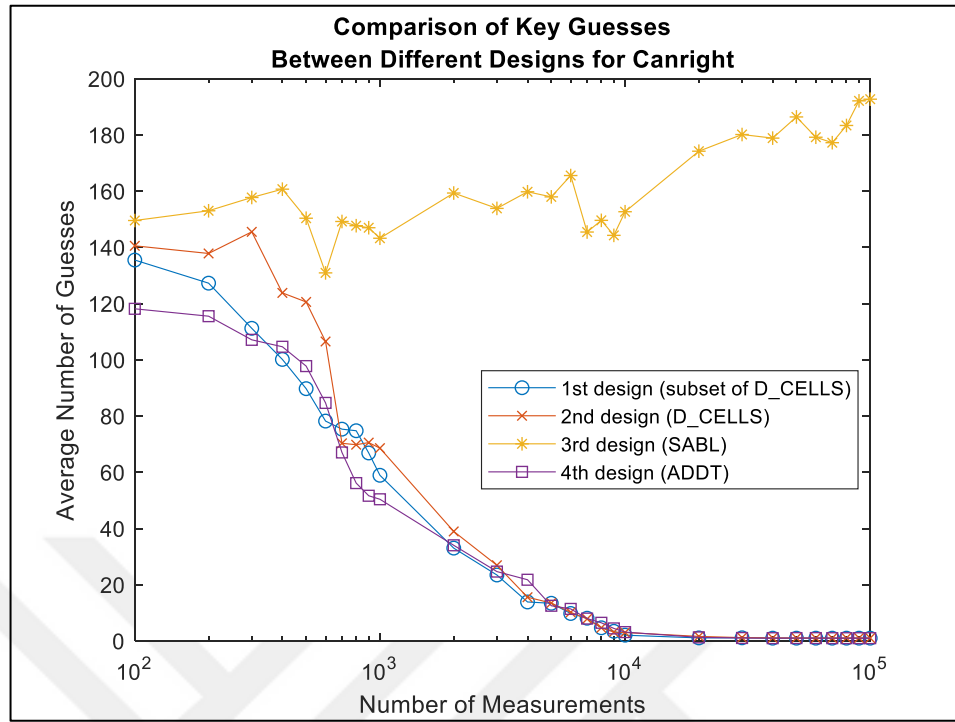


Figure 7.15. Comparison of key guesses between different designs for *Canright*

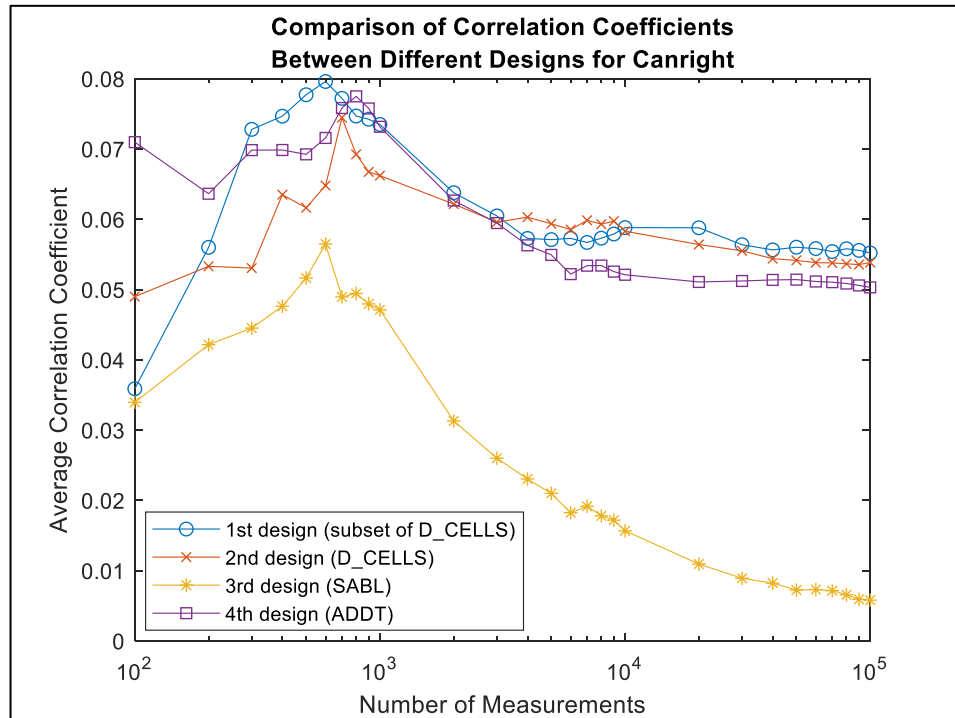


Figure 7.16. Comparison of correlation coefficients between different designs for *Canright*

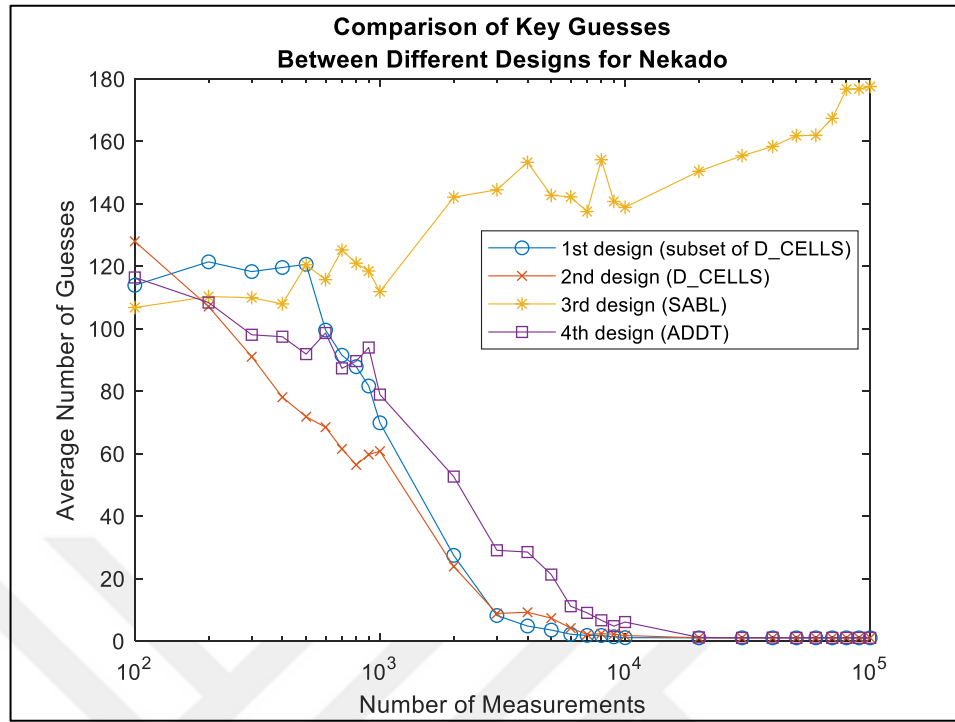


Figure 7.17. Comparison of key guesses between different designs for *Nekado*

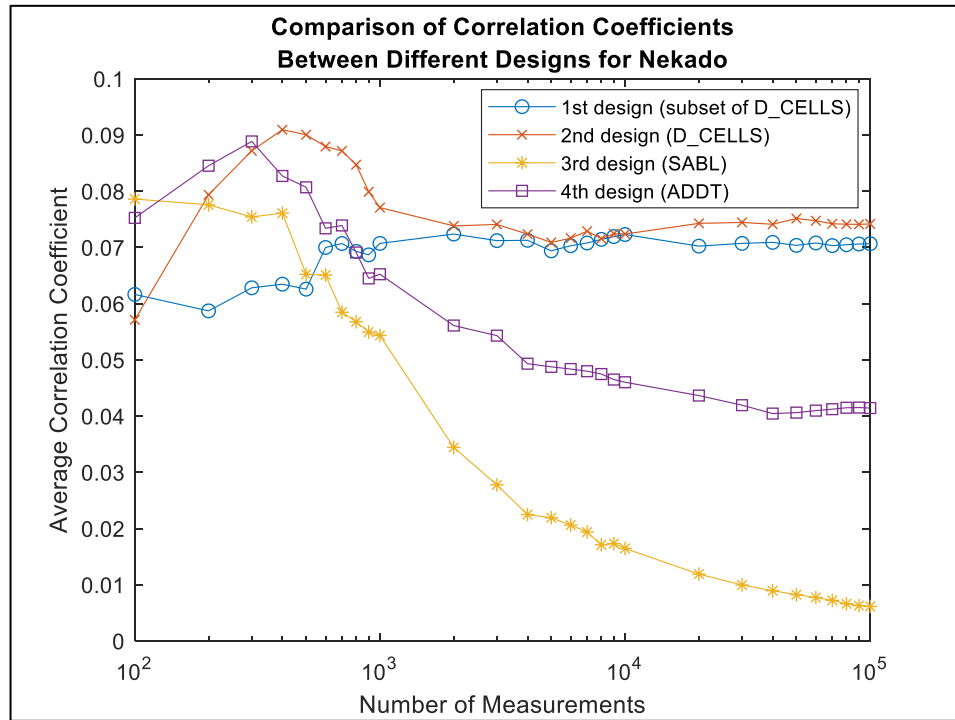


Figure 7.18. Comparison of correlation coefficients between different designs for *Nekado*

7.2.4.5. Results of Template-Based Attack

Before the attack is administered, 256 templates are built for 256 different output values of the series connection of `AddRoundKey` followed by `SubBytes` operation. To this end, mock power measurements of a theoretical sample device with known plaintext and known key were created as described in the previous sections and categorized by their output values such that there are equal number of them for every output value. This number has an influence on the effectiveness of the attack, which will be discussed later. Thereafter, the templates were built by picking five points at the same location of every measurement belonging to each output value. The selection process of these points is described in the third chapter. The covariance and mean matrices of the five points were then calculated for each template to define a multivariate normal distribution that resembles their distribution through different measurements. The completion of this last step brings about 256 covariance and mean matrices, which concludes the template building phase.

The attacking phase of the template-based attack begins by creating mock power measurements just like the correlation coefficient-based attack. But this time, only five designated points are taken out of the 5001 that compose each measurement and the rest are eliminated. Eventually, $p(\text{template}_i|M)$ in equation (3.8) is calculated for every template through the usage of these five points. These posterior probabilities, each of which belongs to an individual template, serve as that template's probability of being the correct one. Since there is a bijection from the set of 256 templates to the set of 256 keys, this in turn is equivalent to guessing the probability of correctness for every key.

Take for instance, a template-based attack on *Boyar_1* (*Boyar* with a limited set of `D_CELLS` gates). As a first step, 256 templates are created for the output of *Boyar_1*. Each template consists of a mean and a covariance matrix. For this example, each matrix is calculated with 10000 measurements (preprocessed as described before) made while the device encrypts uniformly distributed random plaintexts and keys. It is important to note that while the plaintexts and keys for this process are randomly generated, XOR of one specific byte in each plaintext and key has the same output throughout every measurement. This one byte corresponds to the one S-box out of 16 that the templates are being built for. All in all, five points are taken out of each measurement at indices 3416, 4912, 1285, 2096 and 343 for the calculation of the aforementioned matrices. To demonstrate what they look like, the mean

and covariance matrices for the output value E5 in hexadecimal is displayed in (7.6) and (7.7).

$$\mathcal{M} = [3.318 \quad 2.127 \quad 2.038 \quad 2.082 \quad 2.129] \times 10^{-3} \quad (7.6)$$

$$\Sigma = \begin{bmatrix} 0.3846 & 0.2203 & 0.1970 & 0.2209 & 0.2045 \\ 0.2203 & 0.1397 & 0.1289 & 0.1346 & 0.1342 \\ 0.1970 & 0.1289 & 0.1311 & 0.1276 & 0.1363 \\ 0.2209 & 0.1346 & 0.1276 & 0.1348 & 0.1325 \\ 0.2045 & 0.1342 & 0.1363 & 0.1325 & 0.1423 \end{bmatrix} \times 10^{-6} \quad (7.7)$$

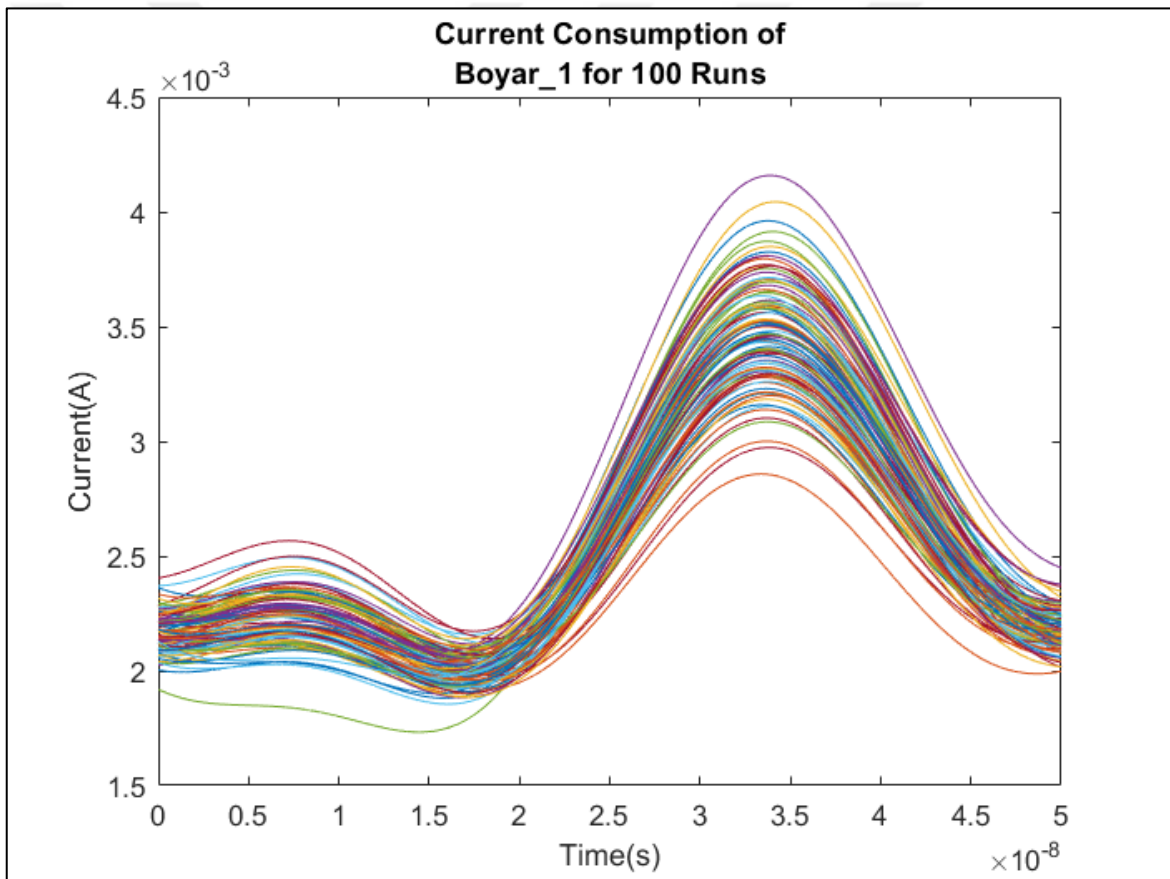


Figure 7.19. Current consumption of *Boyar_1* for 100 runs

During the attacking phase, 100 measurements were created for *Boyar_1* while it encrypts uniformly distributed random known plaintexts with the key in (7.8). These measurements plotted in Figure 7.19 are now thought to belong to the attacked device. Five points at exactly the same indices as the ones used to create the templates, are separated from the rest. The

equation in (3.8) is then applied to this data set of five variables and 100 samples. While doing this, $template_i$ in equation (3.8) is designated by targeting the first byte (53 in hexadecimal) of the key. What is acquired as a result is the probability of each key being the correct one, which is given as a graph in Figure 7.20.

$$\begin{bmatrix} 53 & ED & E4 & 6C \\ 96 & 8F & A7 & F3 \\ E5 & 63 & 65 & 5C \\ 04 & 86 & E2 & EC \end{bmatrix} \quad (7.8)$$

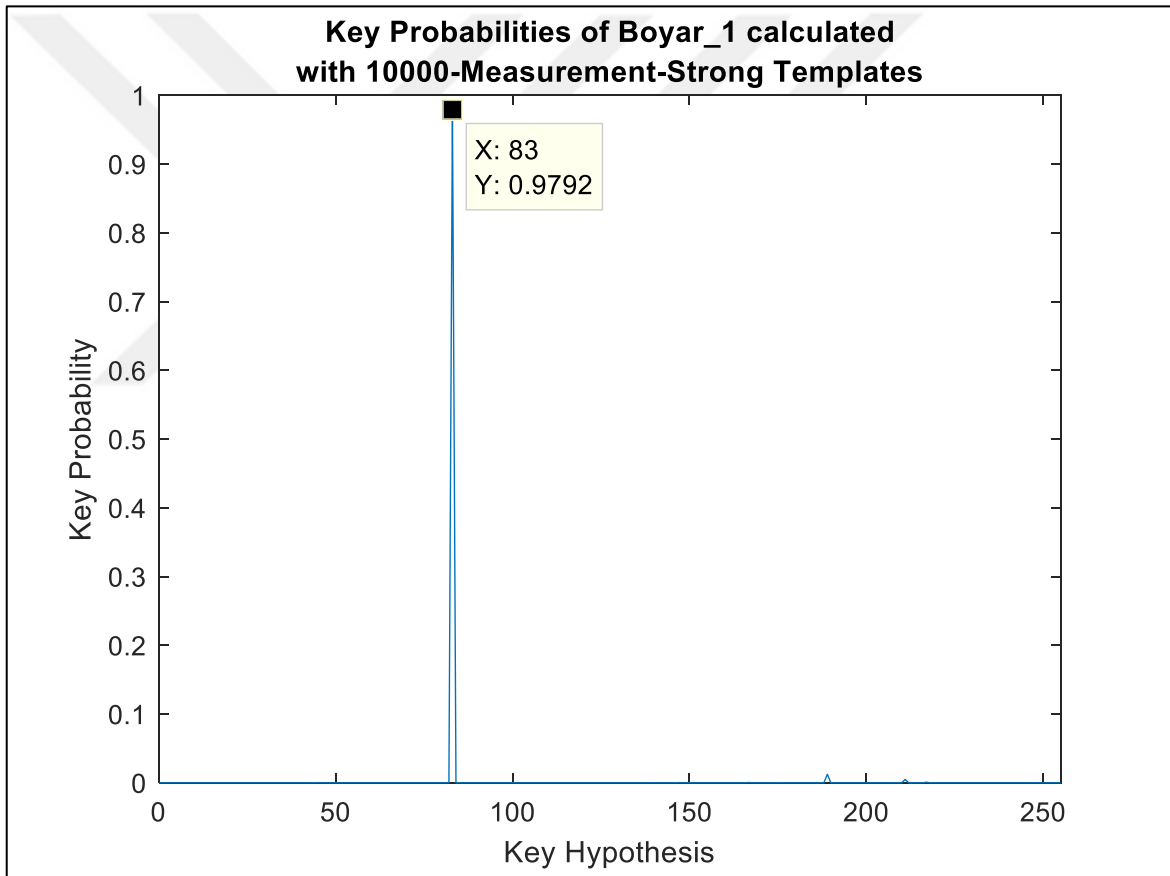


Figure 7.20. Key probabilities of *Boyar_1* via 10000-measurement-strong templates

As proven in Figure 7.20, the attack guesses the probability of the correct key (83 in decimal, 53 in hexadecimal) to be really high (98 percent) compared to the other candidates. Thus, the correct key is extracted successfully with confidence.

Next, in keeping with the analyses involved in the correlation-based attack, the template-based attack is applied to each implementation's first design (*Wolkerstorfer_1*, *Boyar_1*, *Canright_1*, *Nogami_1* and *Nekado_1*) repeatedly. For this purpose, three separate set of templates were created for each circuit. One involving 1000 measurements, another involving 10000 measurements and a last one involving 100000 per template. Subsequently, the circuits were attacked while the number of measurements of the attacked device was swept logarithmically from 10 up to the number used in the template building phase. Testing with more measurements than that would somewhat defeat the objective of the attack, which is taking advantage of a long preparation to shorten the attack. In each case, the results were averaged over 32 attacks. They are given in Figure 7.21, Figure 7.22, Figure 7.23, Figure 7.24, Figure 7.25 and Figure 7.26.

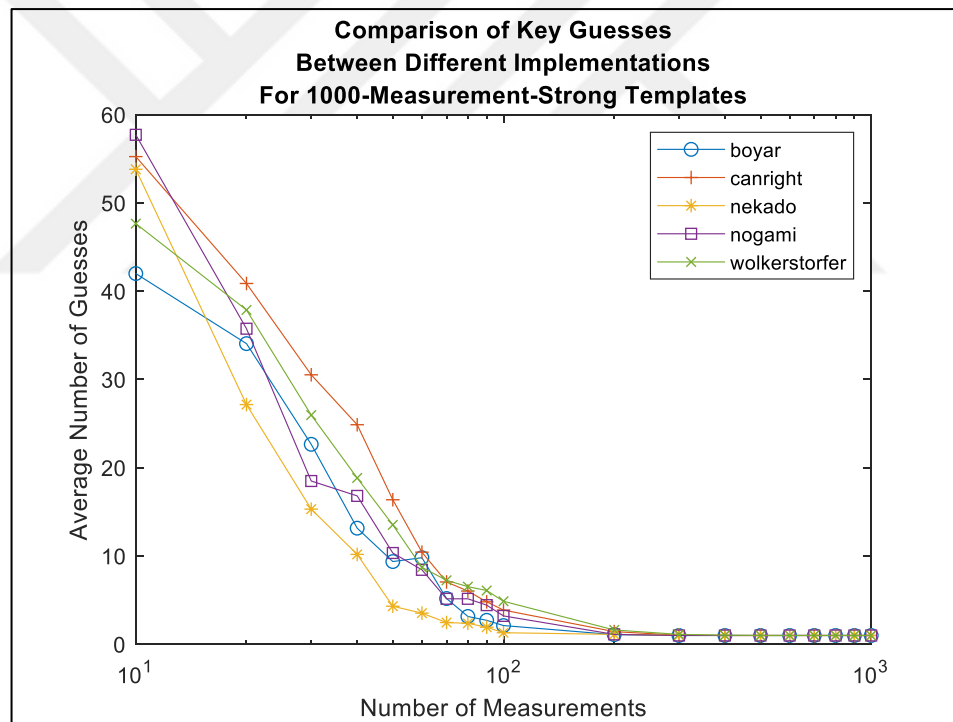


Figure 7.21. Comparison of key guesses between different implementations

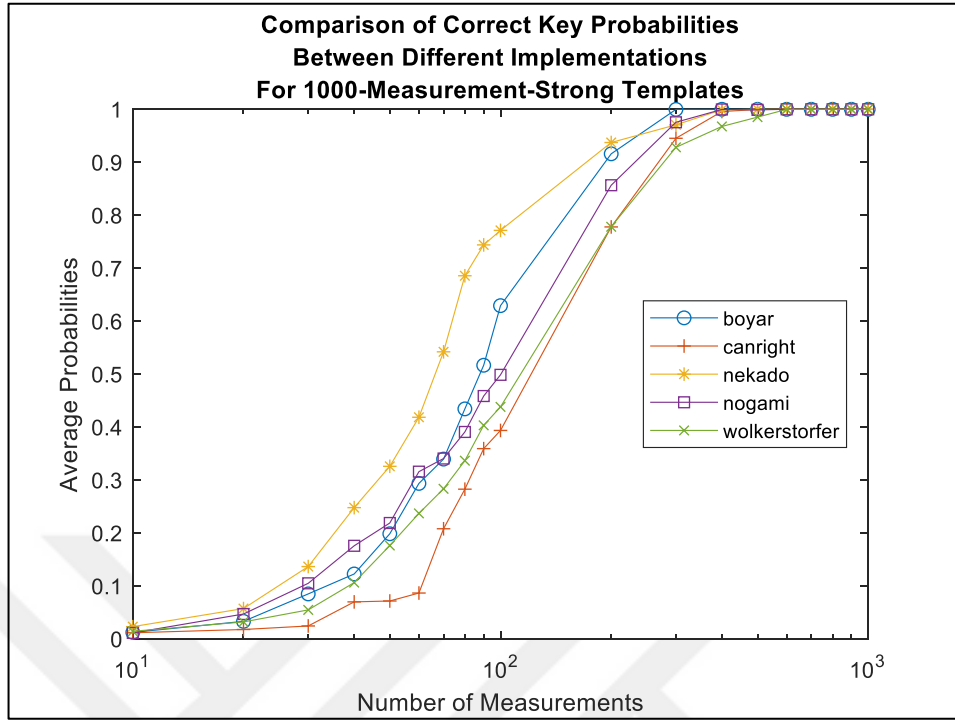


Figure 7.22. Comparison of correct key probabilities between different implementations

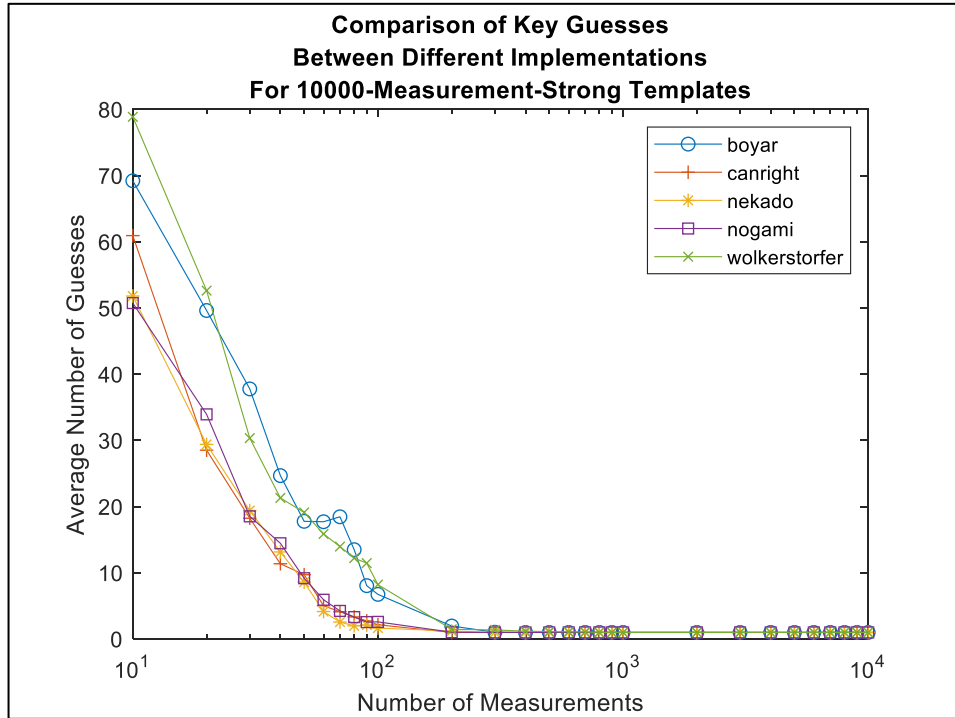


Figure 7.23. Comparison of key guesses between different implementations

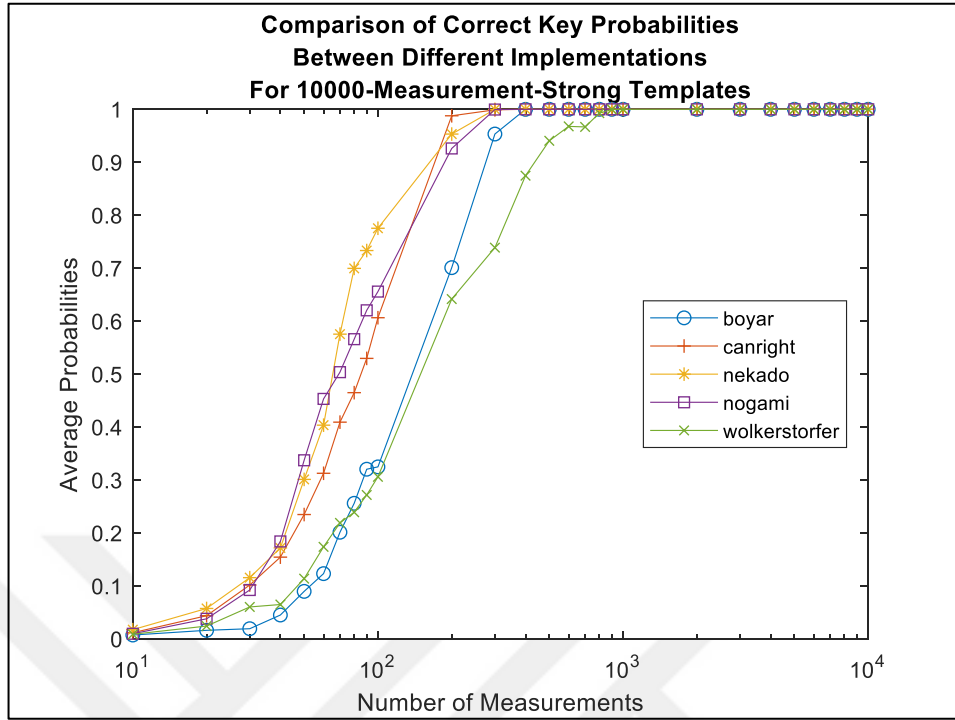


Figure 7.24. Comparison of correct key probabilities between different implementations

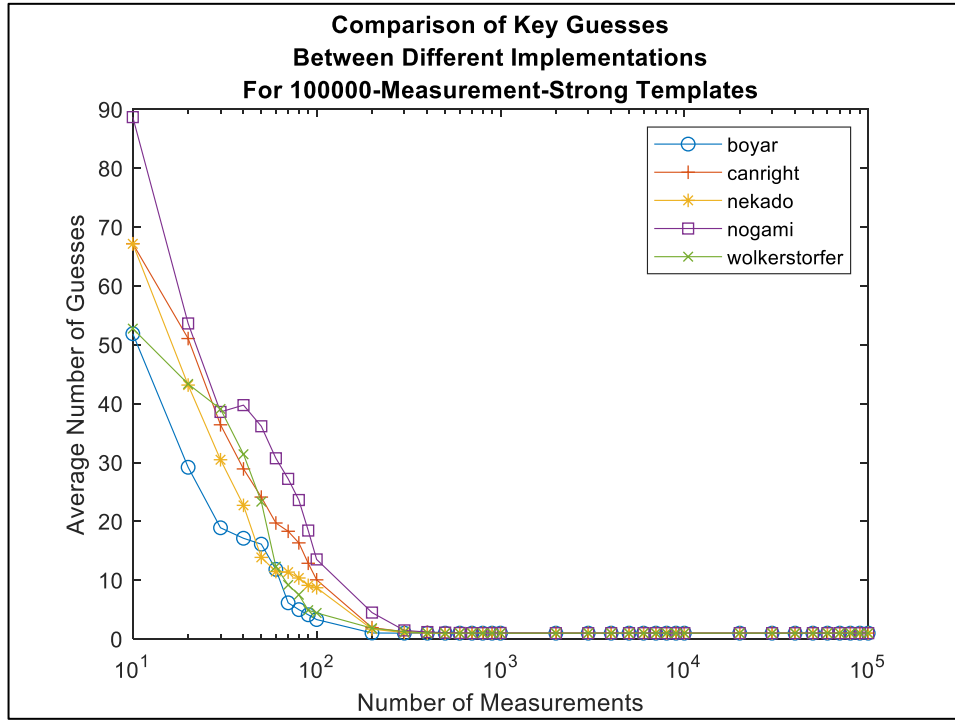


Figure 7.25. Comparison of key guesses between different implementations

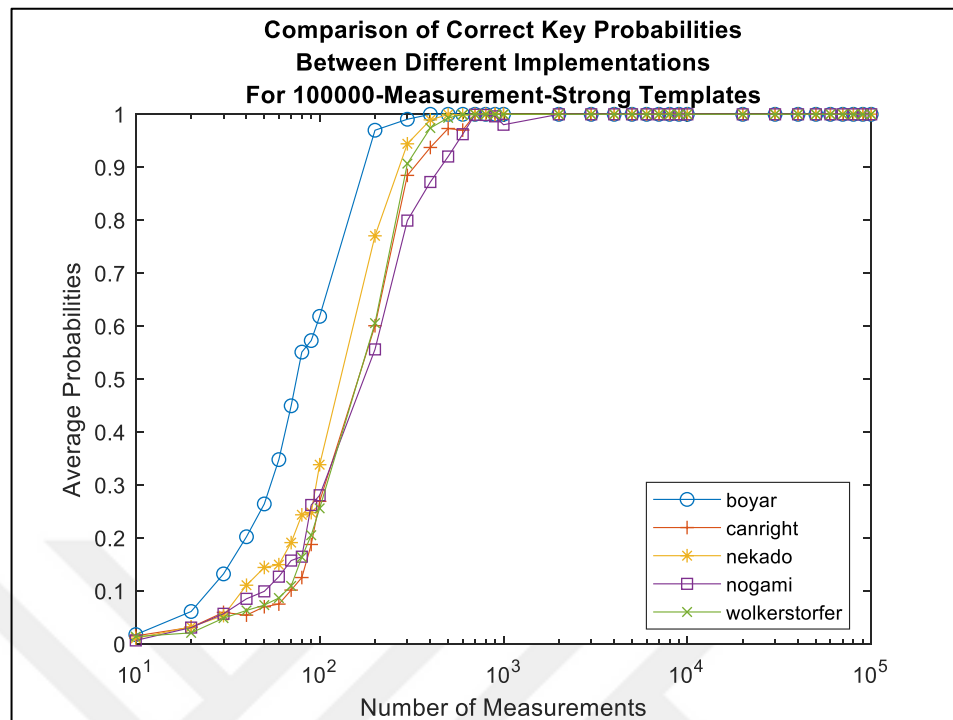


Figure 7.26. Comparison of correct key probabilities between different implementations

Now the comparison is made between different designs, by taking *Canright* and *Nekado* implementations as a basis. The same analyses were applied to these circuits as well and the outcomes are listed in Figure 7.27, Figure 7.28, Figure 7.29, Figure 7.30, Figure 7.31, Figure 7.32, Figure 7.33, Figure 7.34, Figure 7.35, Figure 7.36, Figure 7.37 and Figure 7.38.

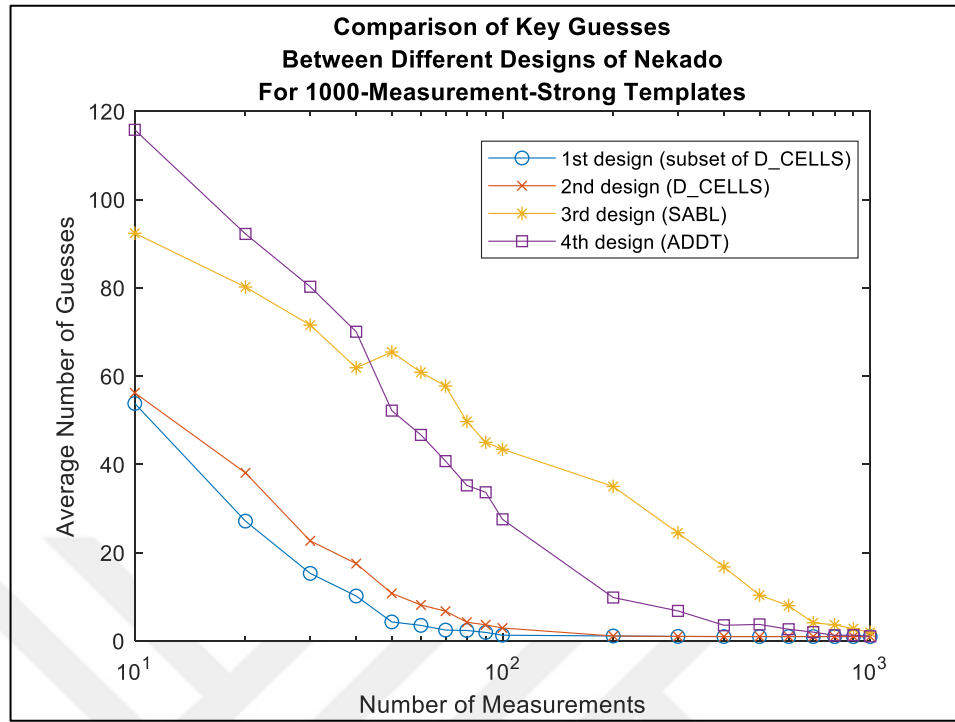


Figure 7.27. Comparison of key guesses between different designs of *Nekado*

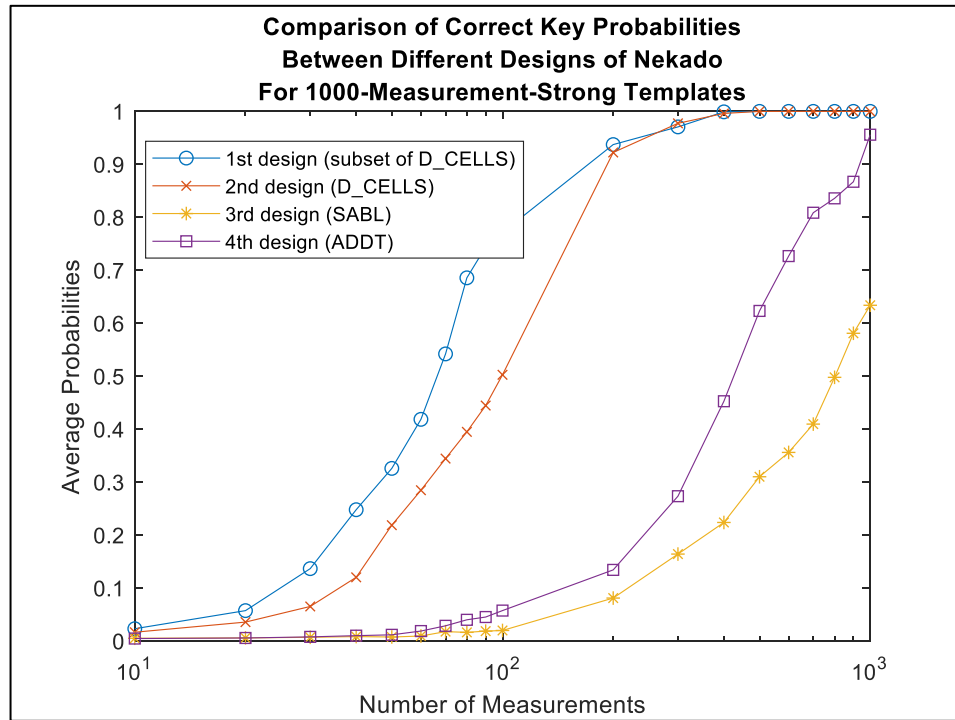


Figure 7.28. Comparison of correct key probabilities between different designs of *Nekado*

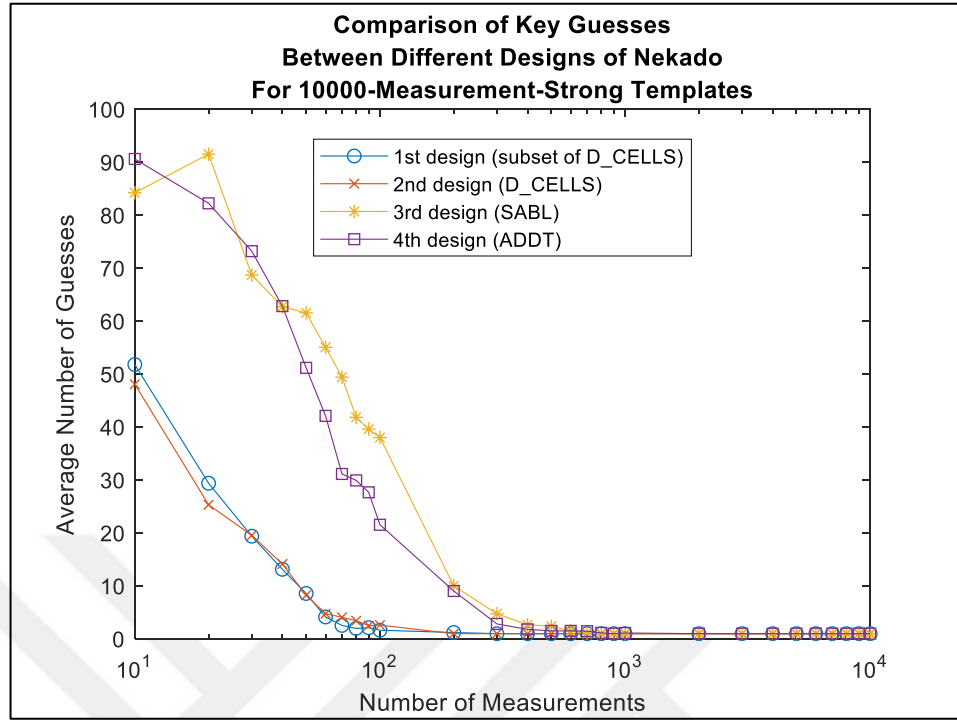


Figure 7.29. Comparison of key guesses between different designs of *Nekado*

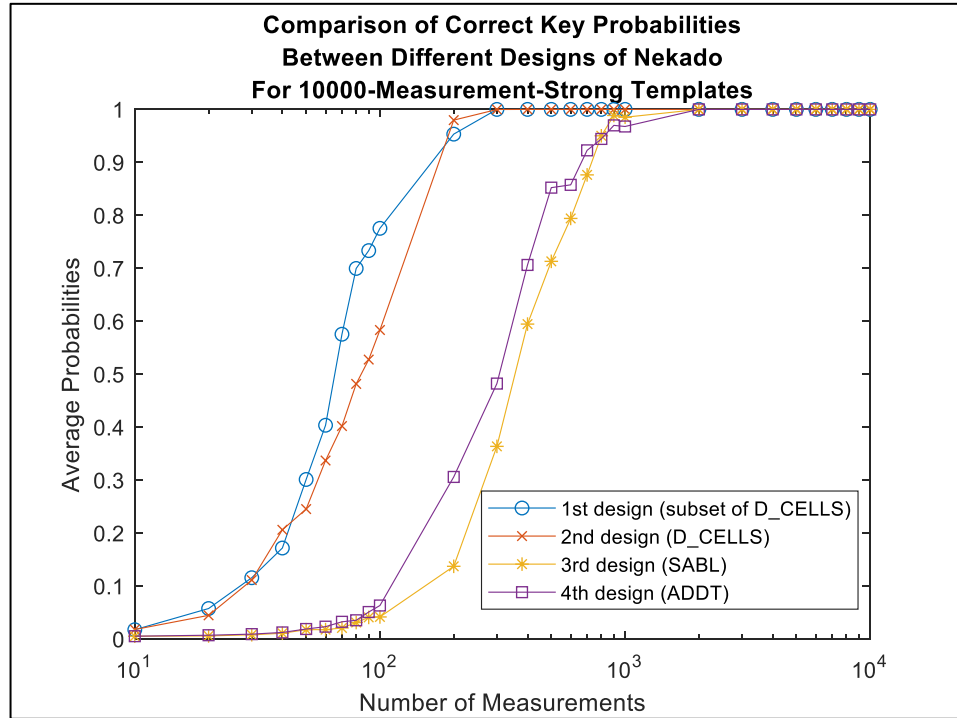


Figure 7.30. Comparison of correct key probabilities between different designs of *Nekado*

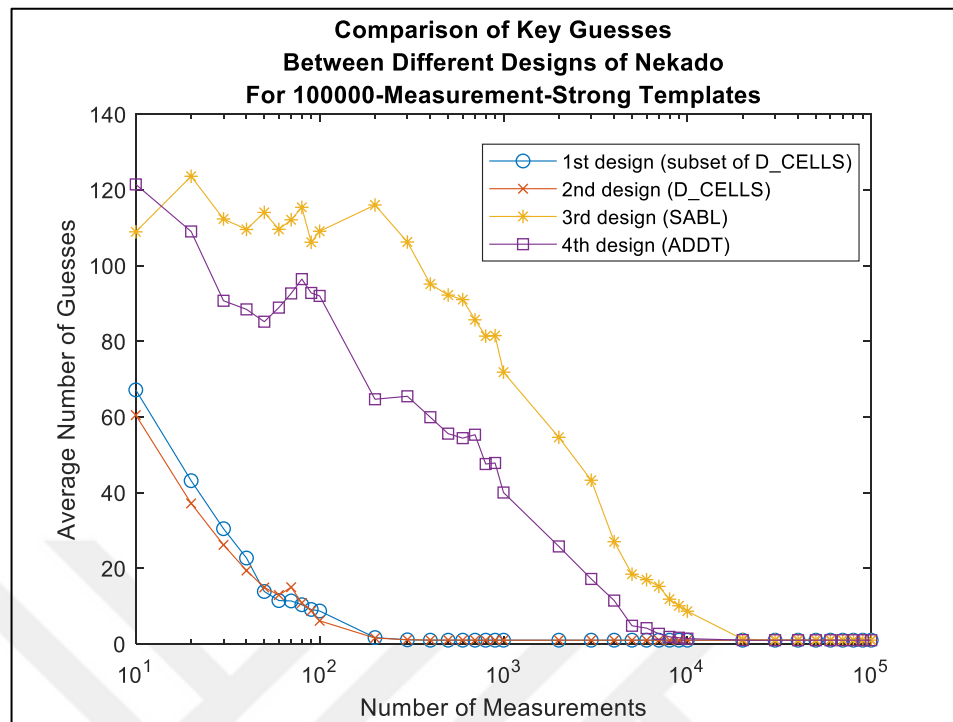


Figure 7.31. Comparison of key guesses between different designs of *Nekado*

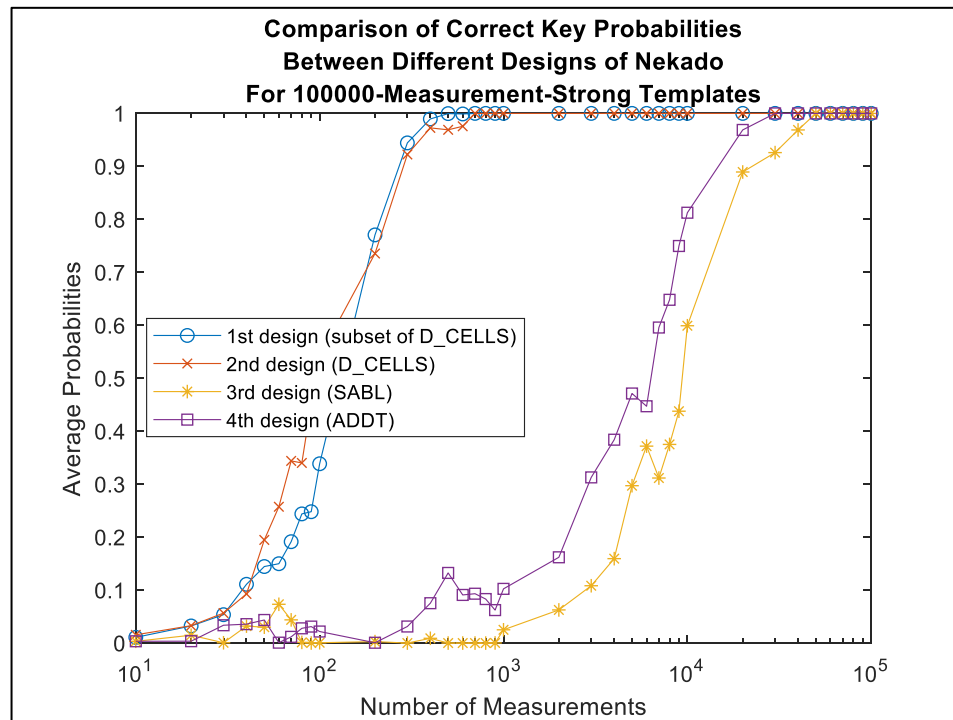


Figure 7.32. Comparison of correct key probabilities between different designs of *Nekado*

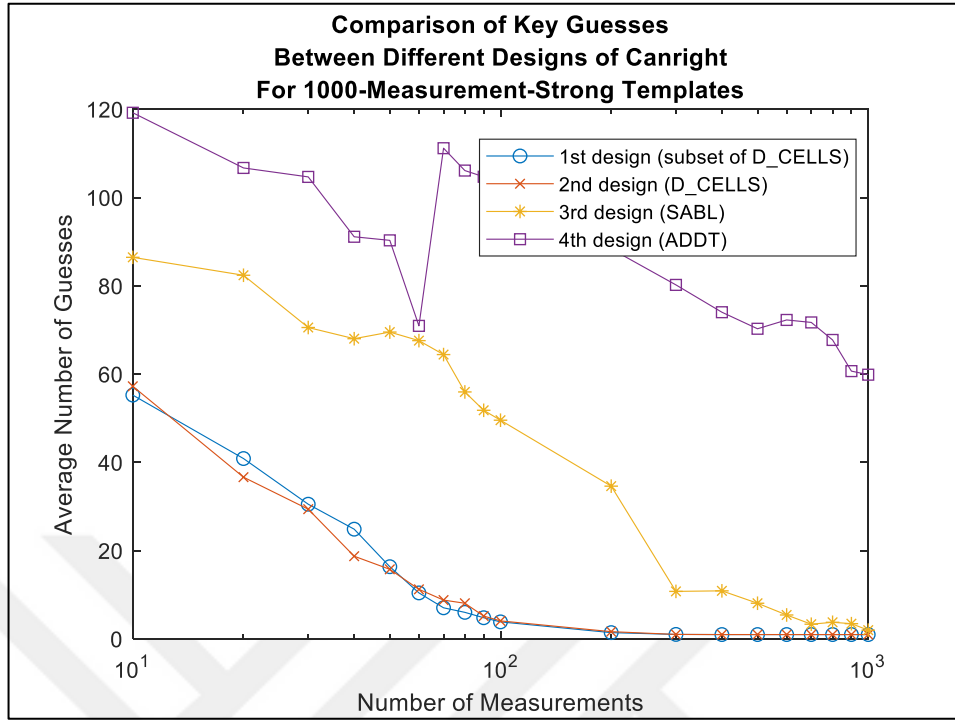


Figure 7.33. Comparison of key guesses between different designs of *Canright*

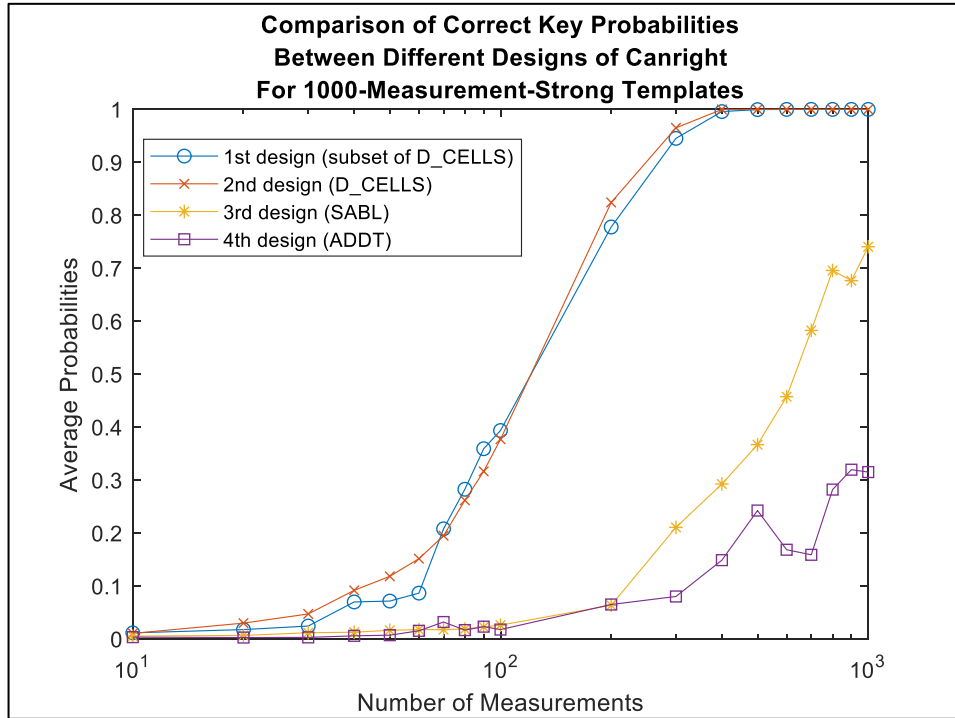


Figure 7.34. Comparison of correct key probabilities between different designs of *Canright*

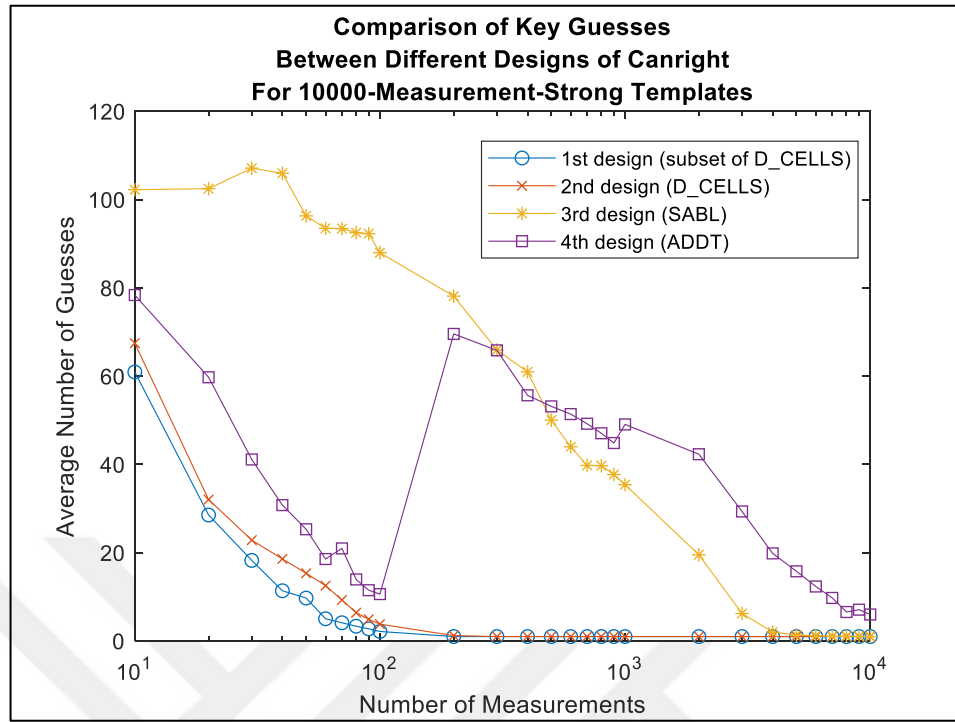


Figure 7.35. Comparison of key guesses between different designs of *Canright*

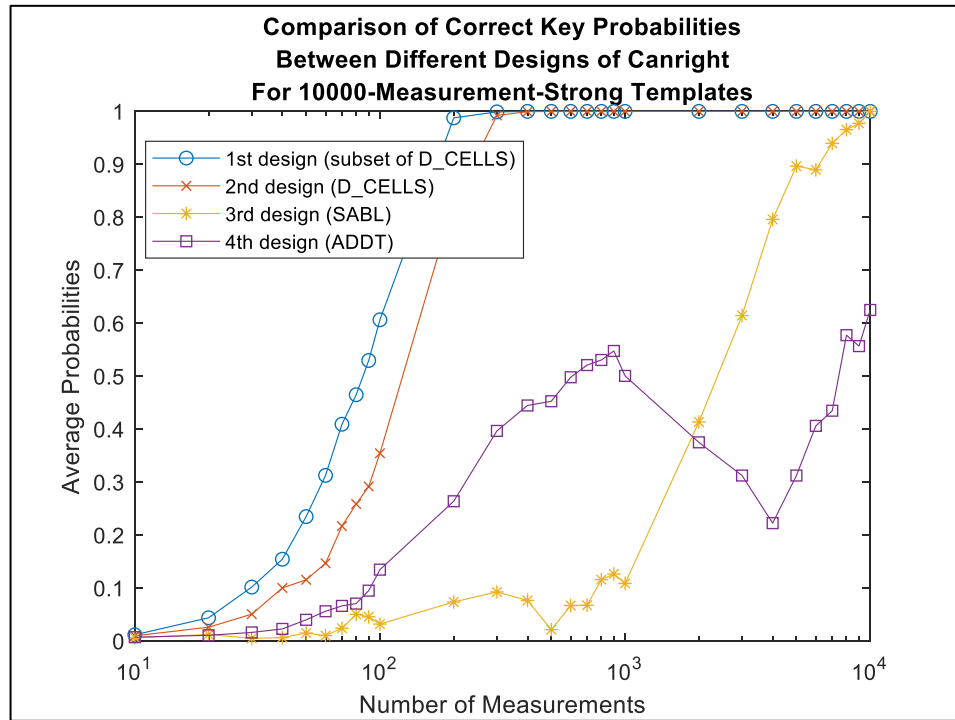


Figure 7.36. Comparison of correct key probabilities between different designs of *Canright*

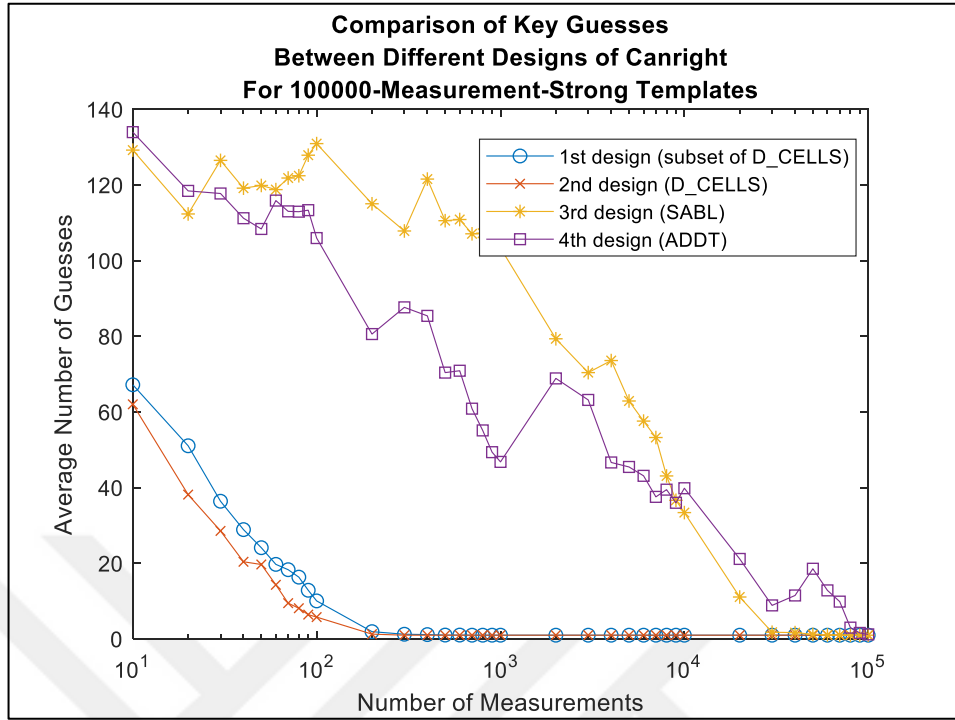


Figure 7.37. Comparison of key guesses between different designs of *Canright*

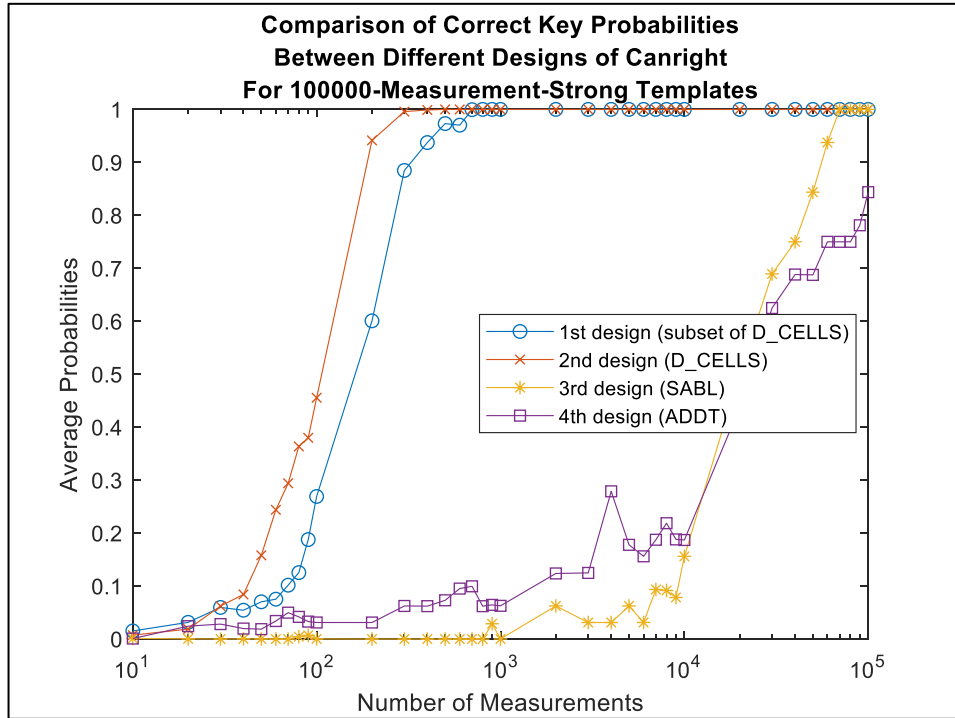


Figure 7.38. Comparison of correct key probabilities between different designs of *Canright*

8. CONCLUSION

The simulated attacks show that dynamic differential gates clearly improve power analysis resistance to the point where the correct key cannot be statistically discerned from the false ones in the case of the correlation coefficient-based method. Yet, the template-based attacks eventually reveal the key.

As perceived in Table 7.12, the RRB method results in the shortest critical path for *Nekado*, *Canright* optimizes the gate count, *Boyar* is well optimized all-around thanks to its employment of great Boolean reduction techniques and *Wolkerstorfer* is the worse one among the five that are examined. Despite all that, the implementation does not make any considerable impact in terms of power analysis resistance when talking about finite field arithmetic implementations, and any miniscule impact that it makes is incidental.

The success of correlation-based attack depends on the hypothetical models which are completely situational. In this project, the initial value of the inputs of the circuits was always zero during each simulated encryption, for which the power consumption was recorded. Consequently, the circuits that follow the first and second design flows virtually consume no power when encrypting the zero input. This makes them very vulnerable against correlation coefficient-based attacks utilizing zero value model. Since the output capacitance of the circuits is not that high, the Hamming weight and bit models do not yield good results. However, by chance, some of the output bits belonging to some of the circuits correlate with the power consumption. This can be observed in Table 7.12 for *Wolkerstorfer_1*. Creating a model that considers those bits jointly may increase the correlation, but it is not guaranteed. While creating complex hypothetical power consumption models, one should also determine what number to assign for each intermediate value. Yet, for an intermediate value with two possibilities this number is irrelevant. So, in a way, in the absence of knowledge, more complex models will likely give worse results and the attacker is left with no choice but to use primitive models. In this context, this thesis proves that zero value model is the only reliable model that can be made to work in most cases.

One unexpected finding of the thesis is that the third design is susceptible to zero value model. On closer inspection, it is seen that, this is caused by the ADDT NAND, AND, NOR and OR gates. To understand the reason, one could inspect the ADDT AND gate in Figure

7.8. The argument holds just as equally even when the inputs and outputs are permuted. During an evaluation phase when only one of the inputs is high and the other one is low, there is only one discharge path to the ground through either **M5** and **M6** or **M7** and **M8**. Contrarily, during an evaluation phase when both of the inputs are high or low, there exist two discharge paths to the ground through all these four transistors. Obviously, the case when both inputs are high or low evaluates faster than the case when one of the inputs is high and the other one is low, on account of **n1** or **n2** discharging faster in the first case. This fact points out to a propagation delay related insecurity in ADDT NAND/AND/NOR/OR.

Furthermore, to makes things worse, no matter how it is implemented finite field arithmetically, the internal calculations of the multiplicative inversion part of `SubBytes` are made up of additions and multiplications of zero by zero. This means the internal logic values are all zeros if they are noninverted or all ones if they are inverted. To connect the dots, this corresponds to all inputs of each gate of *Canright_4* and *Nekado_4* being either one or zero. The ADDT XNOR and XOR are well-balanced internally and do not have this insecurity. The Table 7.12 indicates that there are 98 XNOR/XOR gates and 44 NAND/AND/NOR/OR gates in *Canright_4*, meanwhile there are 106 XNOR/XOR gates and 55 NAND/AND/NOR/OR gates in *Nekado_4*. Evidently, this ratio among NAND/AND/NOR/OR and XNOR/XOR is enough to cause distinct power consumption characteristics as shown in Figure 8.1, where the average power consumption that belongs to zero input and the average power consumption that belongs every other input are plotted separately for *Nekado_4*.

Despite this vulnerability, ADDT performs well in template-based attacks. These attacks also show that increasing the number of measurements used during the profiling phase may reduce the performance of the attack. The jump discontinuities encountered during the attack being performed on *Canright* are collectively another interesting matter that remains unexplained.

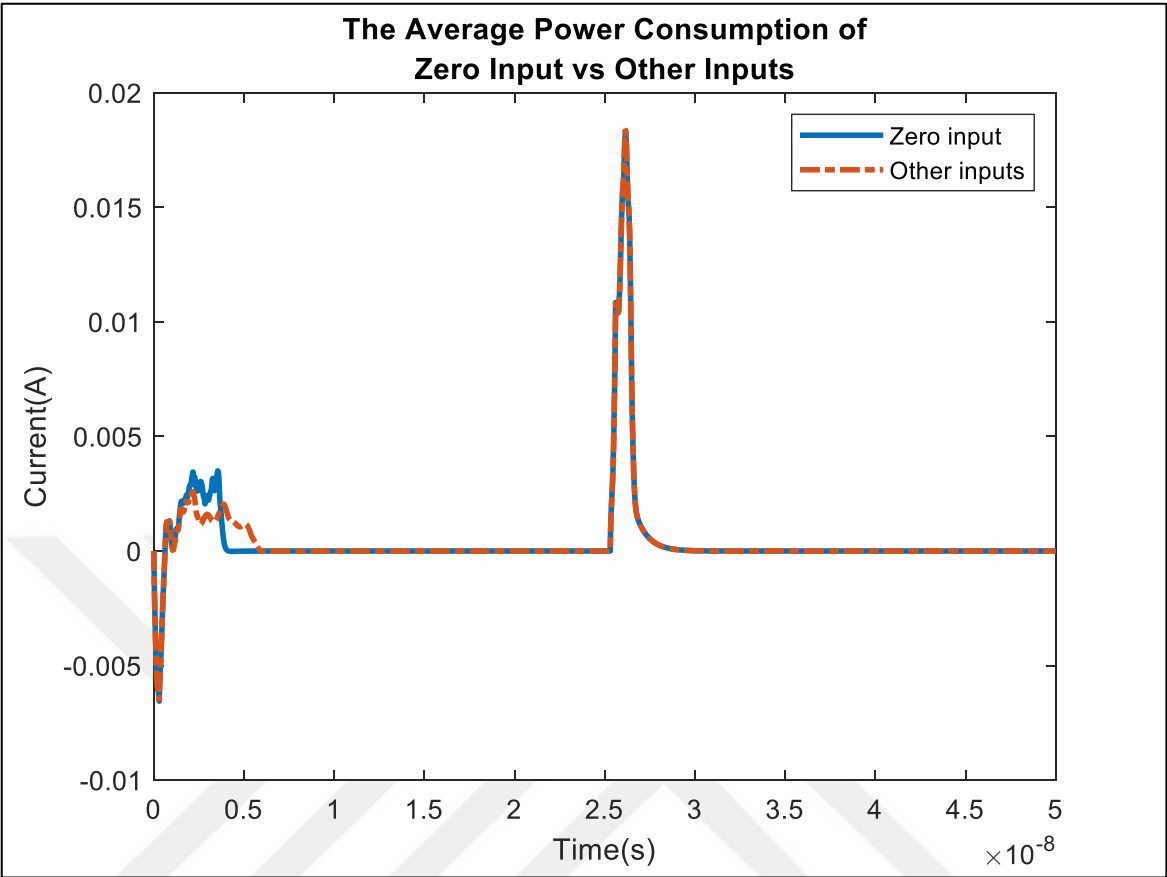


Figure 8.1. The distinctness of power consumption for zero input

REFERENCES

1. Gaines HF. *Cryptanalysis: a study of ciphers and their solution*. New York: Dover Publications; 1956.
2. Liddell HG, Scott R, Jones HS, McKenzie R. *A Greek-English lexicon*. Oxford: Oxford University Press; 1984.
3. Gaines HF. General information. *Cryptanalysis: a study of ciphers and their solution*. New York: Dover Publications; 1956:1-4.
4. Gaines HF. Concealment devices. *Cryptanalysis: a study of ciphers and their solution*. New York: Dover Publications; 1956:4-9.
5. Gaines HF. Transposition types. *Cryptanalysis: a study of ciphers and their solution*. New York: Dover Publications; 1956:9-17.
6. Schneier B. Substitution ciphers and transposition ciphers. *Applied cryptography: protocols, algorithms and source code in C*. New York: John Wiley and Sons, Inc.; 1995:10-3.
7. Gaines HF. Substitution types. *Cryptanalysis: a study of ciphers and their solution*. New York: Dover Publications; 1956:68-9.
8. Luciano D, Prichett G. Cryptology: from caesar ciphers to public-key cryptosystems. *The College Mathematics Journal*. 1987;18(1):2-17.
9. Hill LS. Cryptography in an algebraic alphabet. *The American Mathematical Monthly*. 1929;36(6):306-12.
10. Shannon CE. Communication theory of secrecy systems. *The Bell System Technical Journal*. 1949;28(4):656-715.
11. Kerckhoffs A. La cryptographie militaire. *Journal Des Sciences Militaires*. 1883;9(1):5-38.

12. Kerckhoffs A. La cryptographie militaire. *Journal Des Sciences Militaires*. 1883;9(2):161-91.
13. Berlekamp E, Solomon WG, Thomas MC, Robert GG, James LM, Andrew JV. Claude Elwood Shannon (1916–2001). *Notices of the AMS*. 2002;49(1):8-16.
14. Diffie W, Hellman ME. New directions in cryptography. *IEEE Transactions on Information Theory*. 1976;22(6):644-54.
15. Ferguson N, Schneier B, Kohno T. Diffie-Hellman. *Cryptography engineering: design principles and practical applications*. Indianapolis: Wiley Publishing; 2010:181-95.
16. Schneier B. Data Encryption Standard (DES). *Applied cryptography: protocols, algorithms and source code in C*. New York: John Wiley and Sons, Inc.; 1995:265-303.
17. Diffie W, Hellman ME. Special feature exhaustive cryptanalysis of the NBS Data Encryption Standard. *Computer*. 1977;10(6):74-84.
18. National Institute of Standards and Technology. *Data Encryption Standard (DES)*. Federal Information Processing Standards Publication 46-3; 1999.
19. Menezes AJ, van Oorschot PC, Vanstone SA. DES. *Handbook of applied cryptography*. Boca Raton: CRC Press; 1996:250-9.
20. National Institute of Standards and Technology. Announcing development of a federal information processing standard for Advanced Encryption Standard. *Federal Register*. 1997;62(1):93-4.
21. National Institute of Standards and Technology. Announcing request for candidate algorithm nominations for the Advanced Encryption Standard. *Federal Register*. 1997;62(177):48051-8.
22. National Institute of Standards and Technology. Announcing draft Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard (AES) and request for comments. *Federal Register*. 2001;66(40):12762-3.

23. National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*. Federal Information Processing Standards Publication 197; 2001.
24. Katz J, Lindell Y. Substitution-permutation networks. *Introduction to modern cryptography*. Boca Raton: Chapman and Hall/CRC; 2007:162-70.
25. Preneel B, Rijmen V, Bosselaers A. Recent developments in the design of conventional cryptographic algorithms. *State of the art in applied cryptography, lecture notes in computer science, vol 1528*. Berlin: Springer; 1998:105-30.
26. Daemen J, Rijmen V. AES proposal: Rijndael. 1999.
27. Rijmen V. Efficient implementation of the Rijndael S-box. *Katholieke Universiteit Leuven, Dept. ESAT, Belgium*. 2000.
28. Rudra A, Dubey PK, Jutla CS, Kumar V, Rao JR, Rohatgi P. Efficient Rijndael encryption implementation with composite field arithmetic. *Cryptographic hardware and embedded systems - CHES 2001, lecture notes in computer science, vol 2162*. Berlin: Springer; 2001:171-84.
29. Mathew SK, Sheikh F, Kounavis M, Gueron S, Agarwal A, Hsu SK, et al. 53 Gbps native $GF(2^4)^2$ composite-field AES-encrypt/decrypt accelerator for content-protection in 45 nm high-performance microprocessors. *IEEE Journal of Solid-State Circuits*. 2011;46(4):767-76.
30. Wolkerstorfer J, Oswald E, Lamberger M. An ASIC implementation of the AES SBoxes. *Topics in cryptology, CT-RSA 2002, lecture notes in computer science, vol 2271*. Berlin: Springer; 2002:67-78.
31. Jeon YS, Kim YJ, Lee DH. A compact memory-free architecture for the AES algorithm using resource sharing methods. *Journal of Circuits, Systems and Computers*. 2010;19(5):1109-30.
32. Hodjat A, Verbauwhede I. Area-throughput trade-offs for fully pipelined 30 to 70 Gbits/s AES processors. *IEEE Transactions on Computers*. 2006;55(4):366-72.

33. Zhang X, Parhi KK. High-speed VLSI architectures for the AES algorithm. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2004;12(9):957-67.
34. Rachh RR, Mohan PVA, Anami BS. Efficient implementations for AES encryption and decryption. *Circuits, Systems, and Signal Processing*. 2012;31(5):1765-85.
35. Nikova S, Rijmen V, Schl affer M. Using normal bases for compact hardware implementations of the AES S-Box. *Security and cryptography for networks, SCN 2008, lecture notes in computer science, vol 5229*. Berlin: Springer; 2008:236-45.
36. Zhang X, Parhi KK. On the optimum constructions of composite field for the AES algorithm. *IEEE Transactions on Circuits and Systems II: Express Briefs*. 2006;53(10):1153-7.
37. Satoh A, Morioka S, Takano K, Munetoh S. A compact Rijndael hardware architecture with S-Box optimization. *Advances in cryptology - ASIACRYPT 2001, lecture notes in computer science, vol 2248*. Berlin: Springer; 2001:239-54.
38. Morioka S, Satoh A. An optimized S-Box circuit architecture for low power AES design. *Cryptographic hardware and embedded systems - CHES 2002, lecture notes in computer science, vol 2523*. Berlin: Springer; 2003:172-86.
39. Mentens N, Batina L, Preneel B, Verbauwhede I. A systematic evaluation of compact hardware implementations for the Rijndael S-Box. *Topics in cryptology, CT-RSA 2005, lecture notes in computer science, vol 3376*. Berlin: Springer; 2005:323-33.
40. Kumar S, Sharma VK, Mahapatra KK. Low latency VLSI architecture of S-box for AES encryption. *2013 International Conference on Circuits, Power and Computing Technologies (ICCPCT)*. 2013: IEEE.
41. Wong MM, Wong MLD, Zhang C, Hijazin I. Compact and short critical path finite field inverter for cryptographic S-box. *2015 IEEE International Conference on Digital Signal Processing (DSP)*. 2015: IEEE.
42. Mozaffari-Kermani M, Reyhani-Masoleh A. A low-cost S-box for the Advanced Encryption Standard using normal basis. *2009 IEEE International Conference on Electro/Information Technology*. 2009: IEEE.

43. Canright D. A very compact S-Box for AES. *Cryptographic hardware and embedded systems - CHES 2005, lecture notes in computer science, vol 3659*. Berlin: Springer; 2005:441-55.
44. Canright D. A very compact Rijndael S-box. *Technical report NPS-MA-04-001*. Monterey: Naval Postgraduate School; 2004.
45. Nogami Y, Nekado K, Toyota T, Hongo N, Morikawa Y. Mixed Bases for efficient inversion in $\mathbb{F}((2^2)^2)^2$ and conversion matrices of SubBytes of AES. *Cryptographic hardware and embedded systems, CHES 2010, lecture notes in computer science, vol 6225*. Berlin: Springer; 2010:234-47.
46. Nekado K, Nogami Y, Iokibe K. Very short critical path implementation of AES with direct logic gates. *Advances in information and computer security, IWSEC 2012, lecture notes in computer science, vol 7631*. Berlin: Springer; 2012:51-68.
47. Ueno R, Homma N, Sugawara Y, Nogami Y, Aoki T. Highly efficient $GF(2^8)$ inversion circuit based on redundant GF arithmetic and its application to AES design. *Cryptographic hardware and embedded Systems - CHES 2015, lecture notes in computer science, vol 9293*. Berlin: Springer; 2015:63-80.
48. Mozaffari-Kermani M, Reyhani-Masoleh A. Efficient and high-performance parallel hardware architectures for the AES-GCM. *IEEE Transactions on Computers*. 2012;61(8):1165-78.
49. Batina L, Jakobovic D, Mentens N, Picek S, de la Piedra A, Sisejkovic D. S-box pipelining using genetic algorithms for high-throughput AES implementations: how fast Can we go?. *Progress in cryptology - INDOCRYPT 2014, lecture notes in computer science, vol 8885*. Cham: Springer; 2014:322-37.
50. Dogan A, Ors SB, Saldamli G. Analyzing and comparing the AES architectures for their power consumption. *Journal of Intelligent Manufacturing*. 2014;25(2):263-71.
51. Ueno R, Morioka S, Homma N, Aoki T. A high throughput/gate AES hardware architecture by compressing encryption and decryption datapaths. *Cryptographic*

- hardware and embedded systems – CHES 2016, lecture notes in computer science, vol 9813*. Berlin: Springer; 2016:538-58.
52. Morioka S, Satoh A. A 10-Gbps full-AES crypto design with a twisted BDD S-Box architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2004;12(7):686-91.
 53. Rahimunnisa K, Priya Zach M, Suresh Kumar S, Jayakumar J. Efficient Techniques for the Implementation of AES SubByte and MixColumn Transformations. *Advances in computing and information technology, advances in intelligent systems and computing, vol 176*. Berlin: Springer; 2012:497-506.
 54. Boyar J, Peralta R. A new combinational logic minimization technique with applications to cryptology. *Experimental algorithms, SEA 2010, lecture notes in computer science, vol 6049*. Berlin: Springer; 2010:178-89.
 55. Boyar J, Matthews P, Peralta R. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*. 2013;26(2):280-312.
 56. Wang Z, Zhang X, Wang S, Hao Z, Zheng Z. Application of the composite field in the design of an improved AES S-box based on inversion. *INNOV 2014, The Third International Conference on Communications, Computation, Networks and Technologies*; 2014: IARIA.
 57. Anderson RJ, Bond M, Clulow J, Skorobogatov SP. Cryptographic processors - a survey. *Proceedings of the IEEE*. 2006;94(2):357-69.
 58. Mangard S, Oswald E, Popp T. *Power analysis attacks: revealing the secrets of smart cards*. New York: Springer; 2007.
 59. Kocher P, Jaffe J, Jun B. Differential power analysis. *Advances in cryptology - CRYPTO' 99, lecture notes in computer science, vol 1666*. Berlin: Springer; 1999:388-97.
 60. Kocher PC. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Advances in cryptology - CRYPTO '96, lecture notes in computer science, vol 1109*. Berlin: Springer; 1996:104-13.

61. Dhem JF, Koeune F, Leroux PA, Mestré P, Quisquater JJ, Willems JL. A practical implementation of the timing attack. *Smart card research and applications, CARDIS 1998, lecture notes in computer science, vol 1820*. Berlin: Springer; 2000:167-82.
62. Fahn PN, Pearson PK. IPA: a new class of power attacks. *Cryptographic hardware and embedded systems, CHES 1999, lecture notes in computer science, vol 1717*. Berlin: Springer; 1999:173-86.
63. Örs SB, Oswald E, Preneel B. Power-analysis attacks on an FPGA - first experimental results. *Cryptographic hardware and embedded systems - CHES 2003, lecture notes in computer science, vol 2779*. Berlin: Springer; 2003:35-50.
64. Miller VS. Use of elliptic curves in cryptography. *Advances in cryptology - CRYPTO '85 proceedings, lecture notes in computer science, vol 218*. Berlin: Springer; 1986:417-26.
65. Schramm K, Wollinger T, Paar C. A new class of collision attacks and its application to DES. *Fast software encryption, FSE 2003, lecture notes in computer science, vol 2887*. Berlin: Springer; 2003:206-22.
66. Ferguson N, Schneier B, Kohno T. Birthday attacks. *Cryptography engineering: design principles and practical applications*. Indianapolis: Wiley Publishing; 2010:33-4.
67. Agrawal D, Archambeault B, Rao JR, Rohatgi P. The EM side-channel(s). *Cryptographic hardware and embedded systems - CHES 2002, lecture notes in computer science, vol 2523*. Berlin: Springer; 2003:29-45.
68. Standaert FX, Peeters E, Archambeau C, Quisquater JJ. Towards security limits in side-channel attacks. *Cryptographic hardware and embedded systems - CHES 2006, lecture notes in computer science, vol 4249*. Berlin: Springer; 2006:30-45.
69. Standaert FX, Malkin TG, Yung M. A unified framework for the analysis of side-channel key recovery attacks. *Advances in cryptology - EUROCRYPT 2009, lecture notes in computer science, vol 5479*. Berlin: Springer; 2009:443-61.

70. Chari S, Rao JR, Rohatgi P. Template attacks. *Cryptographic hardware and embedded systems - CHES 2002, lecture notes in computer science, vol 2523*. Berlin: Springer; 2003:13-28.
71. Schindler W, Lemke K, Paar C. A stochastic model for differential side channel cryptanalysis. *Cryptographic hardware and embedded systems - CHES 2005, lecture notes in computer science, vol 3659*. Berlin: Springer; 2005:30-46.
72. Gierlichs B, Lemke-Rust K, Paar C. Templates vs. stochastic methods. *Cryptographic hardware and embedded systems - CHES 2006, lecture notes in computer science, vol 4249*. Berlin: Springer; 2006:15-29.
73. Agrawal D, Rao JR, Rohatgi P. Multi-channel attacks. *Cryptographic hardware and embedded systems - CHES 2003, lecture notes in computer science, vol 2779*. Berlin: Springer; 2003:2-16.
74. Tiri K, Verbauwhecle I. Simulation models for side-channel information leaks. *Proceedings. 42nd Design Automation Conference, 2005*; 2005: IEEE.
75. Kulikowski KJ, Karpovsky MG, Taubin A. Power attacks on secure hardware based on early propagation of data. *12th IEEE International On-Line Testing Symposium (IOLTS'06)*; 2006: IEEE.
76. Elbaz R, Torres L, Sassatelli G, Guillemin P, Anguille C, Buatois C, Rigaud JB. Hardware engines for bus encryption: a survey of existing techniques. *Design, Automation and Test in Europe*; 2005: IEEE.
77. Yu A, Bree DS. A clock-less implementation of the AES resists to power and timing attacks. *International Conference on Information Technology: Coding and Computing, 2004, Proceedings. ITCC 2004*; 2004: IEEE.
78. Yu ZC, Furber SB, Plana LA. An investigation into the security of self-timed circuits. *Ninth International Symposium on Asynchronous Circuits and Systems, 2003, Proceedings*; 2003: IEEE.

79. Guilley S, Hoogvorst P, Mathieu Y, Pacalet R, Provost J. CMOS structures suitable for secured hardware. *Proceedings Design, Automation and Test in Europe Conference and Exhibition*; 2004: IEEE.
80. Gürkaynak F, Oetiker S, Kaeslin H, Felber N, Fichtner W. Improving DPA security by using globally-asynchronous locally-synchronous systems. *Proceedings of the 31st European Solid-State Circuits Conference, 2005, ESSCIRC 2005*; 2005: IEEE.
81. Moore S, Anderson R, Cunningham P, Mullins R, Taylor G. Improving smart card security using self-timed circuits. *Proceedings Eighth International Symposium on Asynchronous Circuits and Systems*; 2002: IEEE.
82. Kulikowski KJ, Ming Su, Smirnov A, Taubin A, Karpovsky MG, MacDonald D. Delay insensitive encoding and power analysis: a balancing act. *11th IEEE International Symposium on Asynchronous Circuits and Systems*; 2005: IEEE.
83. Toprak Z, Leblebici Y. Low-power current mode logic for improved DPA-resistance in embedded systems. *2005 IEEE International Symposium on Circuits and Systems*; 2005: IEEE.
84. Hassan H, Anis M, Elmasry M. MOS current mode circuits: analysis, design, and variability. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*. 2005;13(8):885-98.
85. Mace F, Standaert FX, Hassoune I, Legat JD, Quisquater JJ. A Dynamic Current Mode Logic to counteract power analysis attacks. *The Proceedings of DCIS 2004*; 2004: 186-91.
86. Allam MW, Elmasry MI. Dynamic current mode logic (DyCML): a new low-power high-performance logic style. *IEEE Journal of Solid-State Circuits*. 2001;36(3):550-8.
87. Tiri K, Akmal M, Verbauwhede I. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. *Proceedings of the 28th European Solid-State Circuits Conference*; 2002: IEEE.

88. Nikolic B, Oklobdzija VG, Stojanovic V, Wenyan Jia, James Kar-Shing Chiu, Ming-Tak Leung M. Improved sense-amplifier-based flip-flop: design and measurements. *IEEE Journal of Solid-State Circuits*. 2000;35(6):876-84.
89. Heller L, Griffin W, Davis J, Thoma N. Cascode voltage switch logic: a differential CMOS logic family. *1984 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*; 1984: IEEE.
90. Tiri K, Verbauwhede I. Design method for constant power consumption of differential logic circuits. *Design, Automation and Test in Europe*; 2005: IEEE.
91. Tiri K, Verbauwhede I. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. *Proceedings Design, Automation and Test in Europe Conference and Exhibition*; 2004: IEEE.
92. Sokolov D, Murphy J, Bystrov A, Yakovlev A. Design and analysis of dual-rail circuits for security applications. *IEEE Transactions on Computers*. 2005;54(4):449-60.
93. Aigner M, Mangard S, Menicocci R, Olivieri M, Scotti G, Trifiletti A. A novel CMOS logic style with data independent power consumption. *2005 IEEE International Symposium on Circuits and Systems*. 2005: IEEE.
94. Sundstrom J, Alvandpour A. A comparative analysis of logic styles for secure IC's against DPA attacks. *2005 NORCHIP*; 2005: IEEE.
95. Corsonello P, Perri S, Margala M. A new Charge-Pump based Countermeasure against differential Power Analysis. *2005 6th International Conference on ASIC*; 2005: IEEE.
96. Ratanpal GB, Blalock TN, Williams RD. An on-chip signal suppression countermeasure to power analysis attacks. *IEEE Transactions on Dependable and Secure Computing*. 2004;1(3):179-89.
97. Mesquita D, Techer JD, Torres L, Sassatelli G, Cambon G, Robert M, Moraes F. Current Mask Generation: a transistor level security against DPA attacks. *2005 18th Symposium on Integrated Circuits and Systems Design*; 2005: IEEE.

98. Rakers P, Connell L, Collins T, Russell D. Secure contactless smartcard ASIC with DPA protection. *IEEE Journal of Solid-State Circuits*. 2001;36(3):559-65.
99. Pramstaller N, Gurkaynak FK, Haene S, Kaeslin H, Felber N, Fichtner W. Towards an AES crypto-chip resistant to differential power analysis. *Proceedings of the 30th European Solid-State Circuits Conference*; 2004: IEEE.
100. Herbst C, Oswald E, Mangard S. An AES smart card implementation resistant to power analysis attacks. *Applied cryptography and network security, ACNS 2006, lecture notes in computer science, vol 3989*. Berlin: Springer; 2006:239-52
101. Popp T, Mangard S. Implementation aspects of the DPA-resistant logic style MDPL. *2006 IEEE International Symposium on Circuits and Systems*; 2006: IEEE.
102. Golic JD, Menicocci R. Universal masking on logic gate level. *Electronics Letters*. 2004;40(9):526-8.
103. Schramm K., Paar C. Higher order masking of the AES. *Topics in cryptology, CT-RSA 2006, lecture notes in computer science, vol 3860*. Berlin: Springer; 2006:208-25.
104. Shamir A. How to share a secret. *Communications of the ACM*. 1979;22(11):612-3.
105. Blakley GR. Safeguarding cryptographic keys. *Proceedings of the 1979 AFIPS National Computer Conference*. Monval: AFIPS Press; 1979:313-7.
106. Lidl R, Niederreiter H. *Finite fields (encyclopedia of mathematics and its applications)*. Cambridge: Cambridge University Press; 1996.