Side-Channel-Attack Resistant AES Design Based on Finite Field Construction Variation

A Thesis

Presented in Partial Fulfillment of the Requirements for the Degree
Master of Science in the Graduate School of The Ohio State
University

By

Phillip Shvartsman, B.S.

Graduate Program in Electrical and Computer Engineering

The Ohio State University

2019

Master's Examination Committee:

Xinmiao Zhang, Advisor

Steven Bibyk

© Copyright by
Phillip Shvartsman
2019

Abstract

The Advanced Encryption Standard (AES) is the current standard for symmetric key ciphers and is algorithmically secure. Side channel attacks that target power consumption can reveal the secret key in AES implementations. Masking data with random variables is one of the main methods used to thwart power analysis attacks. Data can be masked with multiple random variables to prevent higher-order attacks at the cost of a large increase in area. This thesis tests the plausibility of using varied finite field construction to prevent power analysis attacks as an alternative to masking. Initially, a design using finite field architecture as the sole countermeasure was investigated. This was followed by varied field construction in conjunction with a low entropy masking scheme. Neither approach provided an acceptable trade off between security and area. Analysis then turned to a combined Boolean and multiplicative masking scheme. Varied construction provided little gain for multiplicative masking. However, varied constructions were found to greatly increase security when used in conjunction with a Boolean random mask. A novel masking scheme for AES resistant to second-order attacks is proposed. Instead of an additional mask, variation in finite field construction is exploited to increase resistance to second-order attacks in Boolean masked shares. As a result, the area requirement is substantially reduced. For an example AES encryption, the proposed design is 12% smaller compared to the previous best design, with a small drop in achievable security level.

Acknowledgments

Thank you to Professor Zhang, for guiding me on this project and providing me with helpful input and encouragement. I am grateful for your patience and everything that you have taught me in the past few years.

Thank you Professor Bibyk, for taking an interest in me when I was an undergraduate. I doubt I would have pursued graduate school if did not have the opportunity to work for you at the start of my junior year.

A special thanks to my family. Mom, Dad, and Emma, I am only where I am because of you. A additional thanks to Emma for proofreading my abstract.

Finally, thank you to my wife Catherine for motivating me as I start each new day and being there for me when I get home.

Vita

2018B.S. Ohio State University

Fields of Study

Major Field: Electrical and Computer Engineering

Table of Contents

	Page
Abstract	ii
Acknowledgments	iii
Vita	iv
List of Tables	vii
List of Figures	viii
1. Introduction	1
2. Background	4
2.1 Finite Field Arithmetic	4
2.2 Advanced Encryption Standard	5
2.3 Varied Finite Field Constructions	8
2.4 Constructing Mapping Matrices	8
2.5 Boolean and Multiplicative Masking	9
2.6 Rotating S-Box Masking	13
2.7 Glitches in Hardware Masking	15
2.8 Simple Power Analysis without Countermeasures	15
2.9 Higher-Order Power Analysis	17
2.10 Test Vector Leakage Assessment	24
3. Varied Finite Field Construction Against Power Analysis	25
3.1 Different Field Constructions as a Countermeasure	25
3.2 Reducing Size of RSM with Different Field Constructions	26
3.3 Proposed Architecture Using Varied Field Constructions to Prevent Second-Order Power Analysis	28

4.	Security Analysis of Proposed Design	33
4.1	Adversary Model	33
4.2	First-Order Security	33
4.3	Security of Masking with Field Construction Variation	34
4.3.1	Security Against HO-DPA	34
4.3.2	Security Against Mutual Information Analysis	37
4.3.3	Simulated Attacks on Varied Construction Masking	39
4.4	Security of Kronecker-Delta Function	41
4.5	Security of Conversion Functions	42
4.6	Security Against Glitch Attack	47
5.	Hardware Overhead Analysis	48
6.	Conclusion and Future Work	51
6.1	Conclusion	51
6.2	Future Work	52
	Appendices	54
A.	Generating Mapping Matrices for Constructions of $\text{GF}(((2^2)^2)^2)$	54
B.	Estimating the PMF for Mutual Information	60
C.	Attack Simulation Code	61
	Bibliography	67

List of Tables

Table	Page
4.1 Summary of simulated attacks	40
5.1 Summary of overhead in gate equivalents and random bits.	49

List of Figures

Figure	Page
2.1 Reduced size Mixcolumns architecture from [1]	6
2.2 Overview of AES with a 128-bit key	7
2.3 Second-order secure conversion circuit implemented in hardware. . . .	11
2.4 Kronecker-Delta function operating on shares.	12
2.5 Overview of RSM scheme.	14
2.6 Result of normal product combining function dependent on x	19
2.7 Result of normal product combining function with x ordered by HW. . . .	20
2.8 Result of absolute difference combining function dependent on x	21
2.9 Result of product combining function dependent on x	21
2.10 Mutual info between shares dependent on x	23
3.1 Overview of proposed modifications to RSM.	27
3.2 Circuit to convert between a three share varied construction scheme and a three share multiplicative scheme.	30
3.3 Required changes to allow for second-order KDF	30
3.4 Extra registers for KD function and SubBytes.	31

3.5	Comparison of area savings between proposed (a) and conventional (b) masking schemes.	31
4.1	Co-variance of leakage of Set 1 with different ordering.	35
4.2	Co-variance of Set 2 leakage compared with	37
4.3	Comparison of mutual information of leakage of Set 1 and Set 2 with standard masked designs.	38
4.4	Mutual information dependent on intermediate value with added noise.	39
4.5	First-order conversion circuit of [2] with each share in a different construction.	42
4.6	Co-variance between multiplicative masked shares with different constructions.	44
4.7	Circuit to convert between a two share varied construction scheme and a three share multiplicative mask.	45
4.8	Second-order leakage when combining output shares of second-order secure KD function.	46

Chapter 1: Introduction

Side-channel-attacks (SCA) are an ever present threat to cryptography circuits. Side-channel-attacks use secondary information of a circuit such as electromagnetic radiation, noise, or power usage. Power analysis is a subset of SCA that has received attention due to the ease of attack. Differential-Power-Analysis (DPA) was proposed in [3] and involves a statistical analysis of many power traces. DPA was followed by more sophisticated attacks such as correlational-power-analysis (CPA) [4] and mutual information analysis (MIA) [5]. For each attack, a model is designed to predict power usage of an intermediate value which depends on a guess of the secret key and some publicly available information. The attacker then compares a point in many recorded power traces where the power is dependent on the intermediate value against the power model. Power usage that reveals information about the intermediate value is referred to as leakage. More statistically rigorous attacks have also been suggested [6], but require the attacker to build a template for each possible value of the intermediate and need a re-programmable copy of the device under attack. This is often unrealistic.

One of the most successful methods of preventing power analysis attacks is masking. Masking splits each intermediate value in a circuit into $d + 1$ shares. The first d shares are random masks and the final share is the combination of the random values with the masks. Recombining the shares at any point in the circuit returns

the actual information being processed. With the device operating on the different shares the power usage becomes uncorrelated to actual intermediate value. With each additional mask, the complexity of the circuit increases greatly. Each share needs a copy of operations, additional registers are needed to store shares, and more random bits need to be generated. An architecture that has $d+1$ shares has d th-order security and must be targeted with a higher-order (HO)-SCA of degree $d + 1$. This attack involves using information at $d + 1$ points on a power trace.

Higher-order attacks becomes exponentially more difficult as d increases due to noise at each point and difficulty in identifying the points of interest [7]. To check security against higher-order attack, each combination of d points are checked for any potential higher-order leakage [8].

Implementation on hardware can leak in unique ways. Using ordinary SCA on combinational logic is challenging because variable timing prevents comparison of a single point across power traces. Often registers are the only intermediate values which can be targeted. [9] describes an attack on glitching logic gate behavior which could potentially be used to recover a key. However, this attack required access to a back-annotated netlist of the exact implementation that was under attack. This assumes a very powerful adversary. In addition, glitches only reveal information when mask and mask data interact [10].

This paper proposes a masking scheme for the Advanced Encryption Standard (AES) that is resistant to second-order attacks but requires only two shares instead of three. AES assumes that each byte is an element of a finite field $\text{GF}(2^8)$ constructed using a chosen irreducible polynomial. There are over 1000 possible constructions for

$\text{GF}(2^8)$. Each construction achieves the same level of security as the original construction and can be mapped to the original construction in 8 unique ways [11]. The two shares are taken as elements in two separate finite field constructions different from the original. This change requires only multiplication with constant 8×8 mapping matrices and small modifications to AES operations. On the contrary, a standard second-order secure implementation requires an additional random mask which invokes a large overhead. Field constructions are chosen which minimize covariance and mutual information between shares. This makes the two most popular higher-order attacks, higher-order DPA (HO-DPA) and multivariate MIA (MMIA), much more difficult. To recover the field constructions adopted, all possible field constructions need to be enumerated which is mathematically intensive. The proposed architecture is also shown to be resistant to glitch attack. The scheme is applied to a byte serial AES encryption and the result is 12% smaller compared to the previous best design in [2].

The thesis is organized in the following manner. Chapter 2 gives background on varied finite field constructions and topics in power analysis. Chapter 3 describes the investigative process of applying varied field construction to prevent power analysis and ends with the proposed masking scheme. Chapter 4 analyzes the security of the proposed scheme. In Chapter 5, a hardware overhead analysis is performed. Chapter 6 concludes the thesis and lists possible future work.

Chapter 2: Background

This chapter gives background information on finite field, AES, previous masking schemes, and power analysis.

2.1 Finite Field Arithmetic

A finite or Galois field is a finite set of elements where the operations addition and multiplication are defined and result in values within the field. In a finite field, there exists an element where addition with any other element results in that element. This is the (00) element of the field. Another element where multiplication with any other element results in that element is the (01) element of the field. The notation for finite fields is $\text{GF}(p^n)$ where GF stands for Galois field, p is the characterization of the field and p^n is the order. Characterization is defined as the smallest integer p where addition of p copies of the identity element result in the zero element. The order is the total number of elements in the field. Within the field, each element α has an order r which is the smallest integer where α^r is equal to the identity element. When an element has the order $p^n - 1$ it is called the primitive element. The finite field $\text{GF}(2)$ contains the elements (00) and (01). Addition and subtraction are defined using the XOR operation and multiplication is defined as the AND operation. In cryptography, Galois fields of $\text{GF}(2^n)$ are used to represent binary data where each

bit is a coefficient of a polynomial. Some polynomial definitions will now be given. A polynomial is irreducible if it has no trivial factors. A primitive polynomial is an irreducible polynomial where the smallest integer s for which the polynomial divides $x^s - 1$ is $p^q - 1$. In the field $\text{GF}(2^n)$, addition remains as the XOR operation and multiplication is done as polynomial multiplication modulo an irreducible polynomial which defines the field [12].

2.2 Advanced Encryption Standard

The Advanced Encryption Standard (AES) is an encryption scheme selected in 2001 by U.S. National Institute of Standards and Technology to be used with electronic data. It was approved for use with federal government data in 2002. AES operates on a 128 bit state and can be used with either an 128, 192, or 256 bit key. The AES state can be represented as a 4-by-4 grid of 16 bytes which is operated on in a series of rounds. A 128 bit key necessitates 10 rounds, a 192 bit key 12 rounds, and a 256 bit key 14 rounds. Each byte in AES is considered an element in the finite field $\text{GF}(2^8)$ defined by the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. To begin, the plaintext and key are added together with the XOR operation which is termed the initial round. The internal state then undergoes the SubBytes, ShiftRows, MixColumns, and the addRoundKey operation which comprise a round. The final round omits the MixColumns step. In the SubBytes operation, each byte of the state is inverted in the finite field and then undergoes an affine transformation. This can be written as $As^{-1} \oplus b$ where s^{-1} is a byte of the state inverted in the finite field, A is a constant 8×8 matrix, b is an 8×1 constant matrix, and $\oplus$ is the XOR operation. ShiftRows

is a shifting of bytes in the state by a constant value dependent on row number. MixColumns takes the four bytes in a column of the state as coefficients of a degree-three polynomial and multiplies the polynomial by $(02)x^3 + (03)x^2 + (01)x + (01)$ modulo $x^4 + 1$. Each value (xx) is an element in the finite field in hexadecimal format. Multiplication of an element x by (03) is equivalent to $(02)x \oplus x$. This allows MixColumns to be computed using only a multiplication by (02) and several sets of XOR gates. Fig. 2.1 shows the MixColumns architecture from [1]. ShiftRows shifts each row by a predetermined offset and does not require any logic gates. The addRoundKey operation is the XOR operation with a roundKey generated by the key schedule [13]. An overview of AES for encryption with a 128-bit key can be seen in Fig. 2.2.

Figure 2.1: Reduced size Mixcolumns architecture from [1]

Figure 2.2: Overview of AES with a 128-bit key

2.3 Varied Finite Field Constructions

Finite field inversion requires a large amount of gates. To reduce required area, inversion is often done in a composite field [14]. A finite field $\text{GF}(2^q)$ is constructed from a degree- q polynomial $p(x)$ which is irreducible over $\text{GF}(2)$. Alternatively a finite field $\text{GF}(2^q)$, can be constructed as $\text{GF}((2^m)^n)$ where $m \times n = q$. Here $\text{GF}(2^m)$ is constructed from a degree- m polynomial $q(x)$ which is irreducible over $\text{GF}(2)$. $\text{GF}((2^m)^n)$ is then constructed from $\text{GF}(2^m)$ using a degree- n polynomial $p_0(x)$ irreducible over $\text{GF}(2^m)$. Values in $\text{GF}(2^q)$ can be isomorphically mapped to $\text{GF}((2^m)^n)$ with a $q \times q$ constant binary matrix. The notation, δ and δ^{-1} , is used in this paper to represent isomorphic mapping matrices and their inverses, respectively. Operations in different constructions are isomorphic. For example $\delta_0 a^{-1} \otimes \delta_0 x \oplus \delta_0 b = \delta_0(a^{-1} \otimes x \oplus b)$. If m is not prime, m can be further decomposed. There are 1020 different possible constructions of a $\text{GF}(256)$ finite field and each construction has 8 unique isomorphic mappings [11]. 720 of the unique constructions are of the field $\text{GF}(((2^2)^2)^2)$. The other AES operations can also be done in a composite field. ShiftRows and AddRoundKey remain unchanged while MixColumns requires the multiplication by (02) and (03) to be mapped to the new field. Multiplication of an element x by $\delta(03)$ is equivalent to $\delta(02)x + \delta x$. The affine transformation of SubBytes also requires mapping to a new composite field.

2.4 Constructing Mapping Matrices

This section will describe how to generate a mapping δ between the finite field used in AES and a composite field construction. A more detailed description and examples can be found in [15]. First an irreducible or primitive polynomial $R(x)$ in $\text{GF}(2^q)$ is

chosen for the standard basis. In AES, this is set as $R(x) = x^8 + x^4 + x^3 + x + 1$. The target composite field is constructed iteratively as described in Section 2.3. The elements (00) and (01) are the same for every construction. To map the (01) element the vector $(0, 0, 0, 0, 0, 0, 0, 1)$ is entered as the right most column of δ . The following relationship must then be satisfied:

$$\delta\beta^i = \alpha^{it}$$

Where β is a root of $R(X)$, α is a primitive element of the composite field, and δ is the mapping matrix. The mapping between the two should be isomorphic in both addition and multiplication. To ensure this, a t value is chosen that satisfies the following equation:

$$R(a^t) = 0$$

Computation of $R(a^t)$ is done using the composite field representation. The next $q-1$ columns are set to a^{it} for $2 < i < q$. For each composite field and standard basis pair there are 8 values of t that result in an isomorphic mapping.

2.5 Boolean and Multiplicative Masking

Masking is a common countermeasure to prevent the power analysis. Masking against simple and higher order attack is described in this section.

The most common type of masking for AES is Boolean masking. In a d th-order secure design, an intermediate value x is split into $d+1$ shares b_0 to b_d . Each share in AES is eight bits. b_0 through b_{d-1} are random masks and $b_d = (\oplus_{i=0}^{d-1} b_i) \oplus x$. $\oplus$ is the XOR operator. The intermediate value can be recovered at any point using $x = \oplus_{i=0}^d b_i$. Linear operations are copied for every share. Non-linear operations, like

inversion in finite field, require combining Boolean shares. When combining shares, additional randomness must be added to retain d th-order security and synchronization registers must be inserted to prevent glitches. [16] [17].

To reduce the complexity of masking finite field inversion, multiplicative masking has been suggested for AES [18]. Let p and q denote multiplicative shares before and after inversion, respectively. p_0 through p_{d-1} are random masks and $p_d = (\otimes_{i=0}^{d-1} p_i) \otimes x$ where $\otimes$ is finite field multiplication. Recovering x becomes $x = \otimes_{i=0}^{d-1} (p_i^{-1}) \otimes p_d$ where p_i^{-1} is the share p_i inverted in the finite field. Recovering x after a multiplicatively masked field inversion requires only one copy of a finite field inversion that operates on the share p_d and no additional randomness or synchronization registers. After inversion, $q_d = p_d^{-1} = (\otimes_{i=0}^{d-1} q_i^{-1}) \otimes x^{-1}$ and x is recovered with $x^{-1} = \otimes_{i=0}^d q_i$.

To utilize both advantages of Boolean and multiplicative masking, [19] has suggested to use Boolean masking and multiplicative masking for linear AES operations and finite field inversion, respectively. Conversions are used that remain d th-order secure. Recently [2], has implemented the combined masking scheme in hardware. Fig. 2.3 shows a second-order secure hardware implementation of the mask conversion circuit. To convert from a Boolean to a multiplicative masked scheme, each Boolean share is multiplied by a new multiplicative mask p_0 and each Boolean share becomes $b_i \otimes p_0$. The total number of shares becomes $d + 2$. The result is synchronized in a register represented by the dotted lines in the figure. Two Boolean shares are then combined and the number of shares becomes $d + 1$. The register prevents the Boolean masks from propagating early and combining before the multiplicative mask has been applied. A Boolean mask is replaced with a multiplicative mask incrementally until there are $d + 1$ multiplicative masks. The share p_d is inverted in

$\text{GF}(2^8)$. After inversion, a new Boolean share b'_i is added to the inversion output and synchronized in a register. A multiplicative mask is removed by multiplying it across all shares. After all multiplicative masks have been removed, the new Boolean shares are $b''_i = (\otimes_i^{d-1} q_i) \otimes b'_i$. The linear affine transform of SubBytes is performed on the new Boolean shares. A first-order secure conversion circuit requires six fewer field multipliers and 11 fewer eight bit registers.

Figure 2.3: Second-order secure conversion circuit implemented in hardware.

The (00) element can not be masked multiplicatively. To address this issue, each (00) element entering a multiplicatively masked section is replaced with a (01) element. Elements (00) and (01) are their own inverse in finite field, and (00) can be restored by adding (01) after inversion. Detection is done with a Kronecker-Delta (KD) function. A KD function computes $\bar{x}_0 \& \bar{x}_1 \& \bar{x}_2 \& \bar{x}_3 \& \bar{x}_4 \& \bar{x}_5 \& \bar{x}_6 \& \bar{x}_7$ where x_i is one bit of x . Special AND gates are required to compute the KD function with masked inputs [17]. These compute across shares and require synchronization registers and extra randomness in each gate to prevent glitches and remain d th-order secure. The

KD function has a latency of three clock cycles. Fig. 2.4 shows a KD function which detects a (00) across two shares.

Figure 2.4: Kronecker-Delta function operating on shares.

The output of a d th-order secure KD function is $d + 1$ bits where XORing the bits together returns a 1 bit if the un-shared value is 0 and 0 otherwise.

Recently, [2] has demonstrated a masking scheme for an AES hardware implementation which uses multiplicative masking for SubBytes and Boolean masking for linear operations. It uses a byte serial architecture and introduces masking conversion circuits in hardware. A KD function using the masked AND gates of [17] prevents (00) element leakage. In addition, the random masks used in the conversion circuit are filtered to remove (00) values. This prevents (00) element leakage from either the mask or the intermediate value.

2.6 Rotating S-Box Masking

The previous masking schemes assume that the masks are random vectors. A different class of low entropy masking mask the intermediate values using a limited set of predetermined masks.

In 2012, a novel masking scheme for AES was proposed in [20]. The scheme is titled Rotating S-Box Masking (RSM). Instead of applying totally random masks, a finite set of 16 masks M_{0-15} is chosen which is resistant to HO-SCA. 16 copies of these initial masks M_j are stored in registers where M_j is equal to M_{0-15} rotated right by j bytes. Before the initial key addition, a j is chosen using a random number generator and M_j is applied. A barrelshifter then rotates the output of the initial key addition j bytes to left so that the masks applied are now in the order of M_{0-15} . The SubBytes step is implemented using 16 look-up tables. The look-up tables perform $S' = \text{SB}(S \oplus M_{0-15}) \oplus \text{rotate}(M_{0-15})$. S is the state, SB is the SubBytes operation, and rotate is rotation by one byte to the right. Using a look-up table removes any possible leakage due to masking, de-masking, or SubBytes. It also allows Boolean masking to be used exclusively as the look-up tables are treated as a linear transformation. Another barrelshifter then rotates the output of the look-up tables right j bytes and the mask applied to the state is now M_{j+1} . The linear operations are then performed on the state. To return the applied mask to M_{j+1} for input into the SubBytes a corrective mask $MM S_{j+1}$ is applied which is equal to $MC(SR(M_{j+1})) \oplus M_{j+1}$ where MC is MixColumns and SR is ShiftRows. $MM S_{j+1}$ is also stored in registers. The final round does not contain MixColumns and a separate corrective mask MS_{j+1} stored in registers removes the M_{j+1} after ShiftRows. RSM uses a round serial design and only stores the output of each round in a register. Hardware glitches are not

considered. Using RSM reduces the required random bits greatly as only the offset j is randomly generated. Fig. 2.5 shows the overall data-path for RSM.

Figure 2.5: Overview of RSM scheme.

2.7 Glitches in Hardware Masking

Hardware masking requires special care. A new type of attack was detailed in [9] which was able to remove masking by exploiting leakages of glitching logic gates. Further analysis in [10] revealed that function which had inputs from both the mask and masked data leaked in way that was dependent on the data. This was due to varied critical paths which allowed the power usage to be dependent on the data when the mask and masked data were combined before the values could be re-masked.

Threshold implementations [21] apply certain rules to prevent this type of leakage. Extra registers are added to synchronize outputs of non-linear functions and re-mask them.

2.8 Simple Power Analysis without Countermeasures

This section describes a first-order attack on a device without masking countermeasures. First, a power model is chosen for an intermediate value x that is dependent on a key guess and some publicly available information. For the AES, a frequent choice for x is a byte of input to the last round's SubBytes. Here $x = \text{SB}^{-1}(k \oplus c)$. c is the publicly available cipher-text, k is a byte of the last roundKey, and SB^{-1} is the inverse SubBytes operator. Attacks use a power model for x , $Z(k_g, c) = \phi(\text{SB}^{-1}(c \oplus k_g))$. k_g is a guess for k and ϕ is commonly the hamming weight (HW) of x or hamming distance (HD) between x and the value of x in the previous clock cycle. For CMOS technology, power usage can be modeled as $p(x) = A\phi(x) + B$, where ϕ is either HW or HD, A is a scaling factor, and B is some noise. The results of the power model are then correlated with a point on the power traces t_1 where the power depends on the intermediate variable. With a correct key guess and power model there will be a

strong correlation between power model and observed power usage. If t_1 is unknown, each point on the power trace can be correlated with the model and the point with the higher correlation is revealed as the point where power is dependent on x . Correlation can be calculated using:

$$\rho(X, Y) = \frac{E[(X - E(X))(Y - E(Y))]}{\sigma_x \sigma_y} = \frac{\text{Cov}(X, Y)}{\sigma_x \sigma_y}$$

Where X and Y are vectors, E is expectation, and σ is standard deviation. Correlation is equivalent to co-variance scaled by each data set's standard deviation. Algorithm 1 describes the CPA attack mathematically.

Algorithm 1 Algorithm for CPA Attack

Input: N power traces T with n points, N corresponding cipher texts C

Output: correctKey

```

1: for  $k_g = 0$  to 255 do
2:   for  $t_i = 0$  to  $n$  do
3:     //If the  $t_i$  is know where the intermediate value is processed, this loop can
       be skipped.
4:     for  $\text{trc}_i = 0$  to  $N$  do
5:        $c = C(\text{trc}_i)$ 
6:        $\text{trace} = T(\text{trc}_i)$ 
7:        $\text{guessPower}(\text{trc}_i) = Z(k_g, c)$ 
8:        $\text{actualPower}(\text{trc}_i) = \text{trace}(t_i)$ 
9:     end for
10:     $\text{corrPoint}(t_i) = \text{Corr}(\text{guessPower}, \text{actualPower})$ 
11:  end for
12:   $\text{corrGuess}(k_g) = \max(\text{corrPoint})$ 
13: end for
14: return correctKey =  $\max(\text{corrGuess})$ 

```

MIA [5] is another type of attack that targets the mutual information between $Z(k_g, c)$ and leakage. Mutual information between two vectors can be calculated with:

$$I(X; Y) = \sum_{X=x, Y=y} p(x, y) \log_2 \left(\frac{p(x, y)}{p(x)p(y)} \right)$$

Mutual information measures the amount of information that an observation of a value of X gives about its corresponding value of Y and is equal to zero when the two are independent. A correct $Z(k_g, c)$ maximizes the mutual information as the two distributions will be identical assuming the power model is correct and there is no noise. MIA requires a probability mass function (PMF) for $L(t_1)$ and $Z(k_g, c)$. $Z(k_g, c)$ returns discrete values with HW or HD and a PMF is easily calculated. Leakages are samples of a continuous distribution and the PMF must be re-constructed. The histogram binning method in [22] is used in this paper. The samples are sorted into histogram bins with the number of bins equal $b_n = 3.49\sigma n^{-1/3}$ assuming Gaussian noise. σ is the standard deviation, and n is the number of samples [22]. Incorrect construction of the PMF can result in an inefficient or impossible attack. MIA is superior to HO-DPA in that it doesn't require assumptions about how the device is leaking information or the masking scheme. Any relationship between the PMFs can be used to recover the key. However, MIA is not as efficient as CPA if HW or HD is known to be the correct leakage model or in the presence of noise. To carry out an MIA attack, Algorithm 1 is modified by adding a line to convert `actualPower` to `actualPowerPMF` and line 10 is replaced with $I(guessPower; actualPowerPMF)$.

2.9 Higher-Order Power Analysis

A higher-order attack must utilize information from $d + 1$ intermediate values to circumvent a masking scheme. Often these are the shares of an intermediate x .

The most straight forward attack, HO-DPA [7], removes masking by exploiting a linear relationship between shares dependent on x and assumes that the data leaks due to hamming weight (HW) or hamming distance (HD). The following analysis assumes an attack on a first-order secure scheme. First, a combining function is used to turn the multi-variate analysis into a uni-variate set. The normalized product combining function is considered the most efficient [23]. It can be written out as $\mathcal{C}(t_1, t_2) = (L(t_1) - E(L(t_1))) \times (L(t_2) - E(L(t_2)))$ for a second-order attack. Where $L(t_i)$ is the power consumption of the share b_i at point t_i of a power trace and E is the expectation. The normal product combining function is equivalent to calculating the co-variance between the two leakages. Simulated leakages for a first-order secure design without noise are calculated using $L(t_0) = HW(b_0) = HW(m)$ and $L(t_1) = HW(b_1) = HW(m \oplus x)$. Where x is constant and m is a random value generated for each leakage pair. The result of the normal product combining function dependent on the value of x can be seen in Fig. 2.6.

Figure 2.6: Result of normal product combining function dependent on x .

For values of x that have a HW of zero and eight there is a large positive and negative co-variance respectively. When HW of x is zero both shares are equal to the random mask. The product combining function always returns a positive value in this case. When HW of x is eight the shares will be inverses of each other and the product combining function will return a negative value. This negative relationship between HW of x and the co-variance of share leakage holds for other values of x . Fig. 2.7 shows Fig. 2.6 with the intermediate value ordered by HW.

Figure 2.7: Result of normal product combining function with x ordered by HW.

The next part of the attack correlates a $Z(k_g, c)$ of x with the result of the normal product combining function. For a correct k_g there will be a high negative correlation when compared to incorrect guesses. Other product combining function exist such as absolute difference and product combining and all attack linear relationship between shares in a similar way [7]. Their respective notation is $\mathcal{C}(t_1, t_2) = |L(t_1) - L(t_2)|$ and $\mathcal{C}(t_1, t_2) = L(t_1) \times L(t_2)$. Fig. 2.8 shows the result of the absolute difference combining function. A correct guess of $Z(k_g, c)$ will have a high positive correlation. Fig. 2.9 shows the result of the product combining function. Similar to the normalized version, a correct $Z(k_g, c)$ will return a large negative correlation. Algorithm 2 describes a generic second-order DPA.

Figure 2.8: Result of absolute difference combining function dependent on x .

Figure 2.9: Result of product combining function dependent on x .

Algorithm 2 Algorithm for HO-DPA Attack

Input: N power traces T with n points, N corresponding cipher texts C

Output: correctKey

```
1: for  $k_g = 0$  to 255 do
2:   //Assumes that  $t_1$  and  $t_2$  are known
3:   for  $\text{trc}_i = 0$  to  $N$  do
4:      $c = C(\text{trc}_i)$ 
5:      $\text{trace} = T(\text{trc}_i)$ 
6:      $\text{guessPower}(\text{trc}_i) = Z(k_g, c)$ 
7:      $\text{combFunc}(\text{trc}_i) = \mathcal{C}(\text{trace})$ 
8:   end for
9:    $\text{corrGuess}(k_g) = \text{Corr}(\text{guessPower}, \text{combFunc})$ 
10: end for
11: return correctKey =  $\max(\text{corrGuess})$ 
```

MMIA [24] is an extension of MIA that computes the mutual information between $Z(k_g, c)$ and the leakages of multiple shares. This can be calculated for a second-order attack using:

$$\begin{aligned} I(L_1; L_2; Z(k_g, c)) &= I(L_1; L_2) - I(L_1; L_2 | Z(k_g, c)) \\ &= \sum_{L_1=x, L_2=y, Z=z} p(x, y, z) \log_2 \left(\frac{p(x, y)p(x, z)p(y, z)}{p(x, y, z)p(x)p(y)p(z)} \right) \end{aligned}$$

The correct $Z(k_g, c)$ minimizes the above expression. This can be explained in the following way. $I(X; Y)$ can be thought of as the mutual information between the leakages $L(t_1)$ and $L(t_2)$. $I(X; Y | Z)$ then calculates the mutual information of the leakages dependent on $Z(k_g, c)$. With a correct $Z(k_g, c)$, $I(X; Y | Z)$ is much larger than the mutual information between shares. As in MIA, PMFs must be constructed for each variable. Fig. 2.10 shows the mutual information of the leakages dependent on the intermediate value.

Figure 2.10: Mutual info between shares dependent on x .

Intermediate dependent mutual information between shares is similar to co-variance. An extreme hamming weight value results in high mutual information as the two leakages become linearly dependent. Mutual information is lowest when the HW of the intermediate value is four. This is because a hamming weight of four is the mode of the distribution. This results in a low amount of information as a hamming weight of four has the most diversity in terms of possible values for each share.

2.10 Test Vector Leakage Assessment

Test Vector Leakage Assessment (TVLA) [25] testing is a standardized process to detect if changing sensitive variables in a circuit results in exploitable leakage. TVLA begins with the null hypothesis that a change in a sensitive variable will not effect leakage. The Welch's T-test is then used to compare two leakage profiles.

$$t = \frac{E(X_1) - E(X_2)}{\frac{\sigma_1^2}{N_1} + \frac{\sigma_2^2}{N_2}}$$

Where E is the expectation, σ is standard deviation, and N_i is the number of samples in profile i . Generally a threshold of 4.5 is chosen for t . A value above 4.5 indicates that the null hypothesis can be rejected with 99% certainty. There are two main categories of leakage detection in TVLA. The general case involves using a fixed sensitive variable for X_1 and a random one for each trace in X_2 . Alternatively, specific values can be targeted to address unique aspects of the masking scheme. A failure of the null-hypothesis does not necessarily guarantee that a key recovery attack is possible. It only pinpoints power usage that is dependent on key.

[26] outlines an approach for applying TVLA testing at the algorithm level. This involves generating power usage for each N intermediate values in a circuit using a chosen leakage model. These will be denoted $c[n]$ where c is a power trace and n is an index of an intermediate value which leaks information. Higher-order leakage can be exposed by pre-processing $c[n]$ using the following equation:

$$c'[n_1 \dots n_k] = \Pi_i^{i=k}(c[n_i] - E(c[n_i]))$$

This amounts to computing the k th-order co-variance between every k possible combinations of intermediate values to test for k th-order security. A $c[n]$ is generated for two different leakage profiles. The Welch's T-test is then performed between the $c[n]$.

Chapter 3: Varied Finite Field Construction Against Power Analysis

This chapter details different attempts to utilize varied field constructions to improve security against power analysis. The final section introduces the proposed architecture as a byproduct of earlier experiments.

3.1 Different Field Constructions as a Countermeasure

Initial, an architecture which used only varied constructions to prevent power analysis was investigated. An AES architecture done in a different field would require a mapping matrix for both the key and plain text and an inverse mapping for the resulting cipher text at the end of encryption. The AES operations would need to be modified to correctly compute in a different finite field. At every point, the intermediate value would now be δx where δ is the mapping matrix to convert an element to the new field construction. In theory, having computation done in different constructions would make the correlation attack in Algorithm 1 more difficult, as $Z(k_g, c)$ is now dependent on δ . However, every construction has the same representation for the elements (00) and (01). This was quickly revealed to be a major source of leakage. To exploit this (00) leakage, $Z(k_g, c)$ can be modified to $\Delta Z(k_g, c)$ where Δ is the Kronecker-Delta Function. With a correct k_g , each value of one in $\Delta Z(k_g, c)$ would

be matched with a very low power usage due to (00). Every value of zero would correlate with some power usage that was greater than zero. A correct key guess would result in a high negative correlation compared with an incorrect key guess. A similar type of attack has been demonstrated on an affine masking scheme in [27]. Even without targeting this zero value leakage, it was found that many choices for δ in $Z(k_g, c)$ resulted in a successful key recovery. There are only nine possible outputs of the hamming weight or distance function. There is a high probability that the values mapped with the δ in $Z(k_g, c)$ will have a similar power output as the values mapped using the δ chosen in hardware.

3.2 Reducing Size of RSM with Different Field Constructions

The previous section explains that choosing a different field construction for AES does not greatly increase security. It was then attempted to apply varied constructions to the RSM scheme of Section 2.6. RSM prevents leakage in SubBytes by using 16 different look-up tables each of which has a large overhead. An architecture was envisioned which used different finite field constructions to implement the SubBytes in logic gates instead. This would reduce the required overhead significantly. Using different constructions was to prevent leakage in each of the SubBytes. The data being processed would now be dependent on the constructions used along with the offset j . The overview of the scheme can be found in Fig. 3.1. Four sets of four different constructions were selected with each set being input into a copy of MixColumns. Grouping into sets of four prevented increased complexity as inputs into Mix Columns must be elements in the same field construction. Using several different constructions did marginally increase resistance to a standard first-order power analysis as the

same element potentially had four different representations. However, for similar reasons illustrated in the previous section, the architecture is ineffective. The (00) element remains unchanged in each construction and the offset j would have little effect on an attack exploiting this (00) element leakage. In addition, masks in RSM are chosen specifically which have low mutual information and prevent higher-order DPA, it would be difficult to preserve these properties with varied constructions as each element would now have four different field construction. Analysis then shifted to more traditional architectures.

Figure 3.1: Overview of proposed modifications to RSM.

3.3 Proposed Architecture Using Varied Field Constructions to Prevent Second-Order Power Analysis

As described in Section 2.9, a higher-order power analysis must use leakage from two different points which ideally leak information about each share. A design is considered where each share is in a different field construction. The leakages would become a function of the field constructions chosen.

The (00) element remains the same between mappings and an attack on the zero element is considered. The exact zero-value isolation attack of Section 3.1 is not possible on a higher order masking scheme as power of one share is independent of the intermediate value. Theoretically, the same attack could be expanded to a higher order by correlating $\Delta Z(k_g, c)$ with the normal product combining function of HO-DPA. This would give a high positive correlation for a correct k_g . The efficacy of the attack still relies on some correlation between HW of x and co-variance of leakages which would be reduced with different constructions.

This idea was initially tested on the architecture in [2]. Experimentation was done to see if different finite field constructions could be used in place of an additional multiplicative random mask in the conversion circuit. The goal was to reduce size. Varied constructions were found to be ineffective for multiplicative masking. This process is given more detail in Section 4.5. The idea was then applied to the Boolean masking portion which led directly to the proposed architecture.

A masking scheme is proposed which uses only two shares for a majority of the data path but is resistant to second-order attacks.

Higher-order attacks target a relationship between multiple shares which is dependent on x . Boolean shares are related linearly. The proposed scheme is similar to

Boolean masking but maps elements in each share to a unique finite field construction. With each share in a different field construction, the relationship between shares is now dependent on the field constructions chosen. There are 1020 composite field constructions of $\text{GF}(2^8)$ each with eight unique mapping matrices. A higher-order attack requires enumerating all possible field constructions which is mathematically intensive. In addition, Section 4 describes how field constructions are chosen to have no linear relationship between share leakage. This makes second-order DPA impossible and MMIA much more difficult. A conventional Boolean masking scheme requires an additional random mask to achieve similar results.

The notations $\mathcal{F}_0$ and $\mathcal{F}_1$ are used for the field construction of share b_0 and b_1 , respectively. The share b_0 remains a random mask, but is treated as $\delta_0 m$ for calculation. m is equivalent to a random mask and δ_0 is the mapping matrix that converts an element from the original construction to $\mathcal{F}_0$. The share b_1 is in the the construction $\mathcal{F}_1$ and is treated as $\delta_1(m \oplus x)$. Each plain text input requires mapping to $\mathcal{F}_1$ and the random mask is applied using $b_1 = \delta_1(x \oplus \delta_0^{-1}b_0)$. The benefits of Boolean masking are retained, linear operations in different field constructions are isomorphic and x can be recovered by $\delta_1^{-1}b_1 \oplus \delta_0^{-1}b_0$. The multiplications by elements (02) and (03) in MixColumns must be modified to compute in different constructions.

A multiplicative masking scheme for SubBytes requires the least amount of gates and is adapted to work with the varied construction masking scheme. This requires a modified KD function and mask conversion circuit. Sections 4.4 and 4.5 show that these non-linear functions do not benefit from varied field constructions. The final design for the conversion circuit can be found in Fig. 3.2. The final version of the KD function is shown in Fig. 3.3. Second-order security for both functions is

guaranteed with an additional random mask added in the register stage before input into these sections. For SubBytes, an additional register is added in the register

Figure 3.2: Circuit to convert between a three share varied construction scheme and a three share multiplicative scheme.

Figure 3.3: Required changes to allow for second-order KDF

stage receiving the SubBytes output to remove the additional random mask before input into the linear functions. Each mask addition and removal is synchronized in registers to prevent hardware glitches. The additional registers required are shown in Fig. 3.4. The masking scheme is implemented on a byte serial architecture with a

Figure 3.4: Extra registers for KD function and SubBytes.

Figure 3.5: Comparison of area savings between proposed (a) and conventional (b) masking schemes.

similar control path as the design in [2]. Two sets of two linear feedback shift registers (LFSRs) store both shares of the key and state array. On each clock cycle the state and key shift. A single SubBytes and KD function are used to process both the state and the key schedule. A byte of the state is input into the KD function three clock cycles before that byte enters the SubBytes. The output of SubBytes is then stored at the end of the LSFR. Every four clock cycles MixColumns is performed on a set group of four registers which make up a column. ShiftRows is done with an alternative shift operation. After SubBytes has been performed on the state matrix, four bytes of the key are then input into the same KD and SubBytes to compute the key schedule. More information about the exact control scheme can be found in [2]. In comparison, a traditional second-order secure implementation requires an extra LSFR to store the extra key and state shares. This area saving is visualized in Fig. 3.5

The extra registers required at the input of the KD function and SubBytes to apply an additional random mask can be implemented by modifying the LFSR registers. This allows the critical path of either the KD function or SubBytes in comparison to [2].

Chapter 5 shows that using only two sets of linear shift registers allows for a great savings in area compared to a conventional second-order secure design.

Chapter 4: Security Analysis of Proposed Design

4.1 Adversary Model

An adversary is considered which can query the device with an unrestricted number of random plain texts and can record the power usage and resulting cipher text. It is assumed that the attacker can not carry out the profiling attacks described in [6]. To break the masking scheme, the attacker will use higher-order SCA. It is assumed that the attacker knows the exact points which leak information about the shares and that leakage can be modeled using:

$$L(x) = \phi(X) + B$$

Where ϕ is either the hamming weight of x or hamming distance between x and the value of x in the previous clock cycle. and B is Gaussian noise.

4.2 First-Order Security

The masking scheme remains first-order secure. A device is first-order secure if the leakage of each share is independent of the x . The XOR operator of a random variable with a uniform distribution and a constant results in a random variable with a uniform distribution. This is the result in the share b_1 in the proposed scheme.

Multiplication with a mapping matrix δ is isomorphic and the added mask remains random and uniform. Thus the share b_1 also remains random, uniform, and resistant to first-order attack.

4.3 Security of Masking with Field Construction Variation

In this section, a possible second-order power analysis attack is considered on Boolean masked shares in different constructions. This type of sharing exists at all points in the proposed architecture except the conversion circuits of the SubBytes operation and the KD function. In a traditional first-order masking scheme, the attacker can attack leakage from the shares $b_0 = m$ and $b_1 = m \oplus x$. Section 2.9 describes how to exploit this leakage using HO-SCA.

In the proposed scheme, the traces now leak the following information: $b_0 = \delta_0 m$ and $b_1 = \delta_1(m \oplus s)$.

4.3.1 Security Against HO-DPA

Security against HO-DPA increases in two ways. Firstly, if the attacker does not know the mapping matrix δ_1 he will be unable to successfully construct the power model $Z(k_g, c)$ for x . Secondly, the correlation between hamming weights and covariance would likely decrease with each share in a different mapping. However, not every construction is equal. Often two chosen constructions have some amount of covariance. This is demonstrated by comparing two sets of mappings in $\text{GF}(((2^2)^2)^2)$.

Set 1 is the construction:

$$p_0 = x^2 + x + 1 \quad p_1(x) = x^2 + x + (10) \quad p_2(x) = x^2 + x + (1000)$$

with the two different valid mapping matrices:

$$\delta_0 = \begin{bmatrix} 10100000 \\ 11011110 \\ 00001100 \\ 01110000 \\ 01101000 \\ 10011100 \\ 00110100 \\ 00000011 \end{bmatrix} \quad \delta_1 = \begin{bmatrix} 10100000 \\ 11011110 \\ 00001100 \\ 01110000 \\ 11001000 \\ 01000010 \\ 00111000 \\ 01110011 \end{bmatrix}$$

Figure 4.1 plots the co-variance between share leakage of Set 1 ordered by HW of x and HW of $\delta_1 x$.

Figure 4.1: Co-variance of leakage of Set 1 with different ordering.

The correlation between $\text{HW}(x)$ and leakage co-variance is zero. If the attacker used a $Z(k_g, c)$ without a mapping matrix, the attack would fail as there would be no relationship between the two. When $Z(k_g, c)$ is modified to take into account δ_1 a clear negative relationship emerges. This would allow a successful key recovery. This also shows that successful key recover could still be possible if δ_1 was incorrect but similar to the correct δ_1 . The 5760 mapping matrices of $\text{GF}(((2^2)^2)^2)$ are used

for $Z(k_g, c)$ and correlated with the co-variance of Set 1. As many as 70% of the mappings result in a correlation above 0.1. The other 30% have a correlation less than 0.1. This shows that choosing arbitrary constructions is not enough.

Instead mappings are chosen which have a zero leakage co-variance for all possible value of x and m . This can be done using Algorithm 3. Calculating zero co-variance in

Algorithm 3 Test if Two Mapping Matrices Have Zero Leakage Co-variance

Input: Mapping matrices δ_0 and δ_1

Output: True or False

```

1: for  $m = 0$  to 255 do
2:    $\text{HW}_{b_0}(m) = \text{HW}(\delta_0 m)$ 
3: end for
4: for  $x = 0$  to 255 do
5:   for  $m = 0$  to 255 do
6:      $\text{HW}_{b_1}(m) = \text{HW}(\delta_1(x \oplus m))$ 
7:   end for
8:   if  $\text{Cov}(\text{HW}_{b_0}, \text{HW}_{b_1}) > 0$  then
9:     return False
10:  end if
11: end for
12: return True

```

the HW leakage model also ensures zero co-variance in the hamming distance model.

In a device that leaks data in hamming distance without noise, leakages are equal to:

$$L(b_0) = \text{HW}(\delta_0(m \oplus m')) = \text{HW}(\delta_0(m''))$$

$$L(b_1) = \text{HW}(\delta_1((m \oplus m') \oplus (x \oplus x'))) = \text{HW}(\delta_1(m'' \oplus x''))$$

This is fundamentally the same as HW leakage.

A mapping in $\text{GF}(((2^2)^2)^2)$ is found to have 1460 other mapping which result in zero co-variance for each combination of x and m . Any mapping can be chosen for either b_0 or b_1 as long as an appropriate partner mapping is chosen. Two constructions

in $\text{GF}(((2^2)^2)^2)$ which have zero leakage co-variance are:

$$p_0 = x^2 + x + 1 \quad p_1(x) = x^2 + x + (10) \quad p_2(x) = x^2 + x + (1000)$$

$$p_0 = x^2 + x + 1 \quad p_1(x) = x^2 + x + (10) \quad p_2(x) = x^2 + (0100)x + (0001)$$

Their corresponding mapping matrices are:

$$\delta_0 = \begin{bmatrix} 10100000 \\ 11011110 \\ 00001100 \\ 01110000 \\ 01101000 \\ 10011100 \\ 00110100 \\ 00000011 \end{bmatrix} \quad \delta_1 = \begin{bmatrix} 10100010 \\ 11010000 \\ 00000010 \\ 10101110 \\ 00010100 \\ 10000100 \\ 00010010 \\ 01110111 \end{bmatrix}$$

Fig. 4.2 compares the co-variance of Set 2 with the co-variance of two shares in a second-order secure design with varied finite field constructions.

Figure 4.2: Co-variance of Set 2 leakage compared with

4.3.2 Security Against Mutual Information Analysis

Reducing co-variance through mapping matrices applies for only a specific leakage model and removes only the linear relationship. Mutual information analysis can

target any relationship between leakages and it would be the preferred attack type against the proposed architecture. All possible mappings $GF(((2^2)^2)^2)$ are tested for low mutual information in a similar way to Algorithm 3. Mutual information is greatly reduced with some mapping combination but there is no mapping pair where mutual information is zero. This makes sense as the relationship between the two leakages is still deterministic, it is just no longer linear. Set 2 is a mapping pair which has a low average mutual information. The average mutual information of Set 1 is higher than Set 2. Figure 4.3 compares the mutual information between leakages of Set 1 and Set 2 against a first-order secure and second-order secure design.

Figure 4.3: Comparison of mutual information of leakage of Set 1 and Set 2 with standard masked designs.

Although mutual information is not equivalent to co-variance there is some relationship between the two. Mutual information peaks occur at similar values as DPA peaks for Set 1.

Power analysis is affected by noise. Mutual information is especially susceptible to noise as the correct PMF must be reconstructed from noisy leakages. The experiment in Fig. 4.3 is repeated with a signal to noise ratio (SNR) of 2. The new leakage function for the shares can be modeled as $L(b_i) = \phi(b_i) + B_i$ where B_i is Gaussian noise. The histogram binning method of [22] is used to reconstruct the PMF. The results can be found in Fig. 4.4.

(a) Set 1 and 2 against 1st-Order Design (b) Set 1 and 2 against 2nd-Order Design

Figure 4.4: Mutual information dependent on intermediate value with added noise.

It can be seen that the mutual information of the leakages in Set 2 approaches a second-order design with added noise.

4.3.3 Simulated Attacks on Varied Construction Masking

To validate the proposed scheme, second-order attacks were performed on simulated HW leakage of the last round's SubBytes input. Simulated leakages of a standard masking scheme are, $L(t_1) = \text{HW}(m)$ and $L(t_2) = \text{HW}(m \oplus \text{SB}(k \oplus c))$. m is the mask, k is a byte of the round key, and c is an output byte of the AES cipher text. The attack power model, $Z(k_g, c)$ is $\text{HW}(\text{SB}(k_g \oplus c))$. In the proposed scheme,

$L_p(t_1) = \text{HW}(\delta_0 m)$ and $L_p(t_2) = \text{HW}(\delta_1(m \oplus \text{SB}(k \oplus c)))$. The attack finds the k_g and δ_1 in $Z(k_g, c) = \text{HW}(\delta_1 \text{SB}(k_g \oplus c))$ that maximizes negative correlation in HO-DPA and minimizes mutual information in MMIA. Security is evaluated using HO-DPA and MMIA with different signal to noise ratios (SNR). Noise is simulated by adding an independent Gaussian noise B to each simulated leakage. The number of leakage measurements for a 90% attack success rate can be found in Table 4.1. The δ_1 in the tables signifies the δ_1 in $Z(k_g, c)$. The proposed architecture entirely prevents

Table 4.1: Summary of simulated attacks

Number of Traces Required to Reveal Secret Key with 90% Success			
SNR	∞	2	1
HO-DPA			
1st Order Secure	150	1300	3000
Set 1, Incorrect δ_1	>1 mil.	>1 mil.	>1 mil.
Set 1, Correct δ_1	650	20000	70000
Set 2, Incorrect δ_1	>1 mil.	>1 mil.	>1 mil.
Set 2, Correct δ_1	>1 mil.	>1 mil.	>1 mil.
2nd Order Secure	>1 mil.	>1 mil.	>1 mil.
MMIA			
1st Order Secure	90	16000	55000
Set 1, Incorrect δ_1	>1 mil.	>1 mil.	>1 mil.
Set 1, Correct δ_1	2500	80000	>1 mil.
Set 2, Incorrect δ_1	>1 mil.	>1 mil.	>1 mil.
Set 2, Correct δ_1	4000	>1 mil.	>1 mil.
2nd Order Secure	>1 mil.	>1 mil.	>1 mil.

HO-DPA on simulated traces even if $Z(k_g, c)$ is known. For MMIA, a correct $Z(k_g, c)$ allows the key to be revealed in a noiseless environment. This is a result of the low

mutual information in Fig. 4.3 that remains even with different constructions. However with added noise, the mutual information is too small to perform a successful attack even with a known $Z(k_g, c)$. In the proposed scheme, hardware glitches would not reduce security, since the shares do not interact [10].

4.4 Security of Kronecker-Delta Function

The effect of varied field constructions on a first-order secure KD function is considered. The proposed architecture uses a KD function to detect and replace the zero element before the multiplicatively masked SubBytes. Masked AND gates are detailed in [17]. A d th-order secure AND gate takes $d + 1$ shares of two bits and outputs $d + 1$ shares of one bit. Each AND gate contains registers to prevent glitches and additional randomness. XORing the output shares returns the AND of the unshared value. The masked AND gates are non-linear and use input from two different shares. Elements in different finite field constructions can not be inputs to the same function. To calculate the shared KD function in the proposed scheme, the different shares must be mapped to the same construction. This would make the input of the KD function vulnerable to second-order attacks.

To secure the KD function without degrading security of the varied construction masking, it is to modified to a second-order secure version. Second-order secure shared AND gates are used which require three input shares. Before the KD function, another mask $\delta_1 m_1$ is XORed to b_1 and the two new shares are stored in separate registers.

The three shares are synchronized to prevent glitches. This is depicted in Fig. 3.4. After the register, the shares b_1 and $\delta_1 m_1$ are mapped to the construction of b_0 and input to the KD function.

4.5 Security of Conversion Functions

This section outlines the different designs considered for the mask conversion function.

Initially, a conversion function was envisioned which used a single mask and varied field construction to prevent second-order attacks. A modified first-order conversion circuit with each share in a different construction was considered. It is shown in Fig. 4.5.

Figure 4.5: First-order conversion circuit of [2] with each share in a different construction.

The multiplicative mask p_0 has its own construction and is converted to the construction of the Boolean shares before multiplication. The Boolean shares are converted to another construction used for inversion before combination. The reverse is done to convert back into Boolean masking. Second-order leakage was considered in the design. Second-order security can be determined by examining leakage at all possible combinations of two points. The leakage from the pairs of points $\delta_0 b_0, \delta_1 b_1$ and $\delta_0 b'_0, \delta_1 b'_1$ has been discussed in the previous section. If the two points do not provide

information of either p_0, b_0 , or b_1 it is assumed that they do not leak information in the second-order. In Fig. 4.5, the only pairs of points that could be attacked are p_0, p_1 and q_0, q_1 . The leakage at these points is, $p_0 = \delta_2 p_0$ and $p_1 = \delta_3 p_0 \otimes x$. Analysis was done with the assumption that the result of the KD function had been successfully applied, and x is never equal to zero. In addition, the masks were considered to be filtered to prevent a (00) element mask from being introduced. This prevents any zero-value leakage of multiplicative masking from being exploited.

Multiplicative masked shares have a different relationship than Boolean masked shares. However, there still remains a linear relationship between the co-variance and the hamming weight of x . Fig. 4.6 shows the co-variance of a multiplicative masked scheme using constructions in Set 1 and Set 2 for δ_2 and δ_3 and a standard first-order secure design.

Figure 4.6: Co-variance between multiplicative masked shares with different constructions.

Computing the correlation between $HW(x)$ with each of the co-variances of Fig. 4.6 results in a correlation greater 0.1. This shows that matrices that satisfy the condition of Algorithm 3 do not satisfy the same condition for multiplicative masking. A search is carried out to find mappings which prevent HO-DPA in multiplicative masking. Line 6 in Algorithm 3 is changed to $HW_{b_1}(x) = HW(\delta_1(x \otimes m))$. No mapping pairs are found which result in zero co-variance. This shows that the varied construction masking in Fig. 4.5 is vulnerable to second-order DPA.

To address this vulnerability, a conversion circuit was considered which used two multiplicative masks. It is shown in Fig. 4.7.

Figure 4.7: Circuit to convert between a two share varied construction scheme and a three share multiplicative mask.

Attacking the same pair of points is now ineffective as the data is split into 3 shares. To check for any other possible leakages in the second-order, the leakage evaluation scheme of [26] was used on the modified conversion circuit. One leakage set was generated with a set intermediate value of the (01) element. This is assumed to be the most likely value to leak information as multiplication with the (01) element results in that element. The other leakage set was generated with a randomized value for x . 1 million leakages are generated for each set and each trace was pre-processed to detect any second-order leakage. The t statistic of the two sets was evaluated and a value above 4.5 did not appear for any pair of points. The null hypothesis that the leakage profiles are independent of intermediate value was accepted.

Critically, this analysis was carried out with the assumption that the output of the KD function had been properly applied. Further consideration revealed that applying the three share output of the second-order secure KD function to the two share varied construction scheme resulted in second-order leakage. To apply the (00) element correction, the three share output would need to be combined into two shares. Two of the shares were XORed and added to b_1 and the remaining share was

added to b_0 . The leakage test of [26] was repeated with the pre-corrected shares and output of the KD function included. It was revealed that the co-variance between the KD output shares after combining was different when x was zero. When x was zero, the two combined shares are a shared (01) and are either (01), (00) or (00), (01) respectively. They have a negative co-variance. For every other possible value of x , they are a shared (00) and are either (00), (00) or (01), (01) and have a positive co-variance. This could be used to reveal a zero valued x with a second-order attack. This leakage is visualized in Fig. 4.8

Figure 4.8: Second-order leakage when combining output shares of second-order secure KD function.

To prevent this leakage, the circuit in Fig. 4.7 was changed to the circuit in Fig. 3.2. The output of the second-order KD function could now be directly added to a three share Boolean scheme without any second-order leakage. However, the circuit

becomes much larger and has the same randomness requirement as the second-order scheme in [2]. The extra registers in Fig. 3.4 also need to be added to the registers at the input and output of SubBytes, further increasing area. This is somewhat offset by the fact that multipliers are now in composite field and are 17% smaller. The test in [26] is repeated a final time and no second-order leakage is detected even when including the input and outputs of Fig. 3.4 and the outputs of the KD function.

4.6 Security Against Glitch Attack

The leakage due to glitches is caused by the mask and mask data interacting or combining. Hardware glitches do not leak information in linear blocks because they are processed for each share separately [10]. In the proposed architecture, shares only interact during recombination in the conversion circuit and in the KD function. The registers in Fig. 3.2 ensure that the multiplicative or Boolean masks are completely applied before shares are recombined. This prevents the recombination from occurring first due to variable critical path. The problem is addressed in the KD function through the registers in the masked AND gates of [17].

Chapter 5: Hardware Overhead Analysis

Hardware complexity of the proposed design is analyzed at the architectural level using an AES encryption with a 128-bit key. The design is compared with the first and second-order secure designs in [2]. Each share of the state and key are read into a 16 eight-bit register array. On each clock cycle the state and key meander through the array. One copy of SubBytes operates on one byte of the state per clock cycle, ShiftRows is done with an alternative meander path, and MixColumns is performed on a column in each share every four clock cycles.

A one-bit register is considered to be three XOR gate equivalents (GE) and an AND gate 0.75 XOR GE. The summary of the comparison can be found in Table 5.1. At the start of encryption, two mapping matrices are needed to perform the construction variation masking on the key and the state. Another two are needed to de-mask the cipher text and map to the original field construction at the end of encryption. Mapping matrices of field constructions of $\text{GF}(((2^2)^2)^2)$ require an average of 16 XOR gates after considering substructure sharing.

The mask conversion in Fig. 3.2 requires four mapping matrices. Overhead is offset by the reduced size of multiplications in the composite field. In the original field construction, a multiplier is 125 XOR GE versus 103 XOR GE in the composite field.

Table 5.1: Summary of overhead in gate equivalents and random bits.

Summary of Overhead in XOR Gate Equivalents			
	1st-Order [2]	2nd-Order [2]	This Paper
Mapping Matrices for Plain Text and Key	0	0	96
SubBytes	753	1815	1661
MixColumns	216	324	320
KDF	126	304	336
Registers	1536	2304	1744
Sum	2631	4747	4158
Percent Overhead 1st-Order [2]	0%	80%	58%
Random Bits	35	85	77

MixColumns in the AES field construction requires four copies of multiplication by (02) each of which is three XOR gates. Multiplication by (03) is done by computing $(02)x \oplus x$ which requires an additional eight XOR gates. MixColumns in a different field requires four copies of multiplication by $\delta(02)$. All possible multiplications by $\delta(02)$ were computed for all field constructions of $\text{GF}(((2^2)^2)^2)$. After substructure sharing, this operation requires on average 16 XOR gates. The architecture in this paper requires two copies of modified MixColumns, one for each share. A standard first-order secure implementation requires two copies of standard MixColumns and a second-order secure requires three. AddRoundKey requires two finite field adders in the first-order design and the proposed design. The second-order design requires an additional finite field adder, for the additional key share.

The KD function in the first-order secure design of [2] requires 126 XOR GE. For the proposed architecture, the second-order KD function is used which has a size of 304 XOR GE. An additional two mapping matrices are required to correctly input into the KD function of the proposed design.

The proposed design uses only two sets of registers for a majority of the data path while the conventional second-order secure requires another set. However, the proposed design needs an additional four eight-bit registers to add an additional mask at the input of the KD function and SubBytes and an additional two are required to receive the three share output of the SubBytes.

The first-order secure design of [2] requires 19 random bits for the non-linear portion. This includes the mask conversion circuit and KD function. 16 bits are required to mask the key and state. The second-order secure design requires 53 for the non-linear portion and 32 bits for the Boolean masking. The proposed design has the same random bit requirement as the second-order design for the non-linear portion. However, it only requires 16 bits for the Boolean shares and 8 bits to secure the KD function and mask conversion.

Overall the proposed design requires 12% fewer logic gates and eight fewer random bits than the second-order design of [2]. The addition of mapping matrices and modification to linear functions is offset by reduction in number of registers and a smaller SubBytes.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

This thesis explores the utility of using varied finite field construction to prevent power analysis attack on AES. Varied constructions as a sole countermeasure was attempted first. This did not provide any increase in security. Varied constructions were then leveraged to reduce size of RSM scheme and multiplicative mask conversion circuits. In both, the security level was reduced greatly and did not justify the reduction in area. For Boolean mask, varied constructions reduced the area of a second-order secure design while retaining a similar security level.

This thesis proposes a second-order power analysis attack resistant design which uses varied finite field construction instead of an addition mask for second-order security. Exploiting varied finite field construction only requires insertion of constant 8×8 mapping matrices along the data path and small modifications to AES operations. Using an additional random mask would require another set of registers to hold the shared key and state array in addition to extra copies of linear operations. Comparing leakages of shares with different possible constructions in Chapter 4 reveals that properly chosen constructions can eliminate co-variance between leakages. This can entirely prevent a second-order DPA attack and severely impacts the efficacy of

MMIA. However, varied constructions are not as effective for multiplicative masking and can not properly secure the Kronecker-Delta function. These portions need to use an additional random mask. Chapter 5.1 shows that overall the proposed architecture is 12% smaller than a conventional second-order secure design and requires 8 fewer random bits.

6.2 Future Work

The analysis in this paper heavily assumes that the device will leak as a function of either hamming weight or hamming distance. In practice, it is unlikely that the devices will perfectly leak in this regard. There could be some other relationship that is introduced which could be targeted by MMIA or HO-DPA.

In addition, template attacks are not discussed. Template attacks also do not require any assumptions about the leakage function. They can succeed if there is some set of points on the power trace which are related with the key and each other. A template attack can be expanded to a large amount of points with little effort. Template attacks can pinpoint and attack leakages which are introduced through the specific implementation. Both these possibilities point to the need to implement the proposed architecture on hardware and address new issues that arise.

The analysis also only used the current best combining function which was specifically designed to target Boolean masked shares. Some other combining function may exist which would be more efficient to target the proposed design. However, the proposed design removes linear relationship between share leakage and the combining function would probably be significantly more complicated. It would also need to be a function of the field constructions used, further increasing complexity. The

large number of possible constructions and lack of linear relationship point to large degree of security against a theoretical combining function but the possibility should be explored.

Varied masking may also have a similar effect on next order security when combined with higher-order masking. This would need further investigation.

Appendix A: Generating Mapping Matrices for Constructions of $\text{GF}(((2^2)^2)^2)$

This Matlab code can be used to generate all mapping matrices of the composite field $\text{GF}(((2^2)^2)^2)$. The code requires the following functions to quickly convert between binary and decimal notation:

```
1 function y = d2b(x,m)
2 % Number of divisions necessary ( rounding up the log2(x) )
3 c = max(0,floor(log2(x))) + 1;
4 if nargin<2
5 m=c;
6 end
7 shift = m-c;
8 y(m) = 0; % Initialize output array
9 for i = 1:c
10 r = floor(x / 2);
11 y(shift+c+1-i) = x - 2*r;
12 x = r;
13 end
```

```
1 function y = b2d(x)
2 % Convert a binary array to a decimal number
3 %
4 % Similar to bin2dec but works with arrays
5 % instead of strings and is found to be
6 % rather faster
7 z = 2.^(length(x)-1:-1:0);
8 y = sum(x.*z);
9 end
```


The main function is given below. It tries every possible coefficient for the primitive polynomials and tries to build a mapping matrix with that polynomial. The format of the polynomials is:

$$p_0 = x^2 + x + 1 \quad p_1(x) = x^2 + (P1) * x + (P0) \quad p_2(x) = x^2 + (L1)x + (L0)$$

It stores the mapping matrices in the variable T_array, and it used in other code by loading it with the Matlab load command.

```

1 %findTArrays.m
2 clc;clear;close all;
3 T_array = zeros(8,8);
4 %Primitive Poly for AES
5 R = [8 4 3 1 0];
6 %Store Primitive Polynomial Coefficients
7 set = zeros(1,4,1);
8 count = 0;
9 T_array = {};
10 for P1 = 1:3
11     for P0 = 1:3
12         for L1 = 1:15
13             for L0 = 1:15
14                 T = buildTArray(8, R, [222 P1 P0 L1 L0]);
15                 if sum(T) > 0
16                     count = count + 1;
17                     T_array(count) = {T};
18                     if (size(T,3)<8)
19                         disp([P1 P0 L1 L0]);
20                     end
21                     set(:, :, count)=[P1 P0 L1 L0];
22                 end
23             end
24         end
25     end
26 end
27 save('T_array.mat', 'T_array');
```

This function evaluates $R(\beta)$ for every possible value of β if it equals zero a mapping matrix is built with that β .

```

1 function [ T_array ] = buildTArray(num, R, construction)
2 %Output a T array 8*num large
3 %based off of R primitive polynomial and
4 %P1,P2,L1,L2
5 type = construction(1);
6 if type == 222
7     P1 = d2b(construction(2),2);
8     P0 = d2b(construction(3),2);
9     L1 = d2b(construction(4),4);
10    L0 = d2b(construction(5),4);
11 end
12
13 T_array = zeros(8,8,1);
14 count = 0;
15 for i = 0:1:255
16     beta = flip1r(d2b(i,8));
17     if type == 222
18         out = evalElementMapping222(R,beta,L1,L0,P1,P0);
19     end
20     if bi2de(out) == 0
21         count = count + 1;
22         if type == 222
23             T = buildIsomorphicMapping222(beta,L1,L0,P1,P0);
24         end
25         T_array(:, :, count) = T;
26     end
27     if count == num
28         break;
29     end
30 end
31 end

```

The function to evaluate $R(\beta)$ is given next.

```

1 function [ out ] = evalElementMapping222(R,beta,L1,L2,P1,P2)
2 %Test if R(beta) = 0 where R is
3 %the polynomial for the field GF(2^8) and
4 %beta is an element in GF2222.
5 out = [0 0 0 0 0 0 0 0];
6 for i=1:numel(R)
7     buf = powgf2222(beta,R(i),L1,L2,P1,P2);
8     out = xor(buf,out);

```

```

9 end
10 end

```

The functions next are different operations in finite field needed to find mapping matrices.

```

1 function [ c ] = powgf2222( a,pow,L1,L2,P1,P2 )
2 %
3 if pow == 0
4     c = [0 0 0 0 0 0 0 1];
5 elseif pow == 1
6     c = a;
7 elseif pow > 1
8     c = a;
9     for i = 1:pow-1
10         c = multgf2222(a,c,L1,L2,P1,P2);
11     end
12 end
13 end

```

```

1 function [ c ] = multgf2222( a,b,L1,L0,P1,P0 )
2 %Multiply a,b mod the two primitive
3 %polynomials defined by L1,L0,P1,P0
4 a_1 = a(1:4);
5 a_0 = a(5:8);
6 b_1 = b(1:4);
7 b_0 = b(5:8);
8
9 a_1b_1 = multgf222(a_1,b_1,P1,P0);
10 if L0 == L1
11     a_1b_1L0 = multgf222(a_1b_1,L0,P1,P0);
12     a_1b_1L1 = a_1b_1L0;
13 else
14     a_1b_1L0 = multgf222(a_1b_1,L0,P1,P0);
15     a_1b_1L1 = multgf222(a_1b_1,L1,P1,P0);
16 end
17 a_0b_0 = multgf222(a_0,b_0,P1,P0);
18 a_1b_0 = multgf222(a_1,b_0,P1,P0);
19 a_0b_1 = multgf222(a_0,b_1,P1,P0);
20
21 c(1:4) = xor(xor(a_1b_0,a_0b_1),a_1b_1L1);
22 c(5:8) = xor(a_1b_1L0,a_0b_0);

```

23 | end

```
1 function [ c ] = multgf222( a,b,P1,P2 )
2 %Mult a,b mod x^2+P1x + P2
3 a_1 = a(1:2);
4 a_2 = a(3:4);
5 b_1 = b(1:2);
6 b_2 = b(3:4);
7
8 a_1b_1 = multgf22(a_1,b_1);
9 if P2 == P1
10     a_1b_1P2 = multgf22(a_1b_1,P2);
11     a_1b_1P1 = a_1b_1P2;
12 else
13     a_1b_1P2 = multgf22(a_1b_1,P2);
14     a_1b_1P1 = multgf22(a_1b_1,P1);
15 end
16 a_2b_2 = multgf22(a_2,b_2);
17 a_1b_2 = multgf22(a_1,b_2);
18 a_2b_1 = multgf22(a_2,b_1);
19
20 c(1:2) = xor(xor(a_1b_2,a_2b_1),a_1b_1P1);
21 c(3:4) = xor(a_1b_1P2,a_2b_2);
22 end
```

```
1 function [ c ] = multgf22( a,b )
2 %Multiply a,b modulo PP = x^2+x+1
3 if isequal(a,[0 0])
4     c = logical([0 0]);
5     return
6 elseif isequal(b,[0 0])
7     c = logical([0 0]);
8     return;
9 elseif isequal(a,[0 1])
10     c = logical(b);
11     return;
12 elseif isequal(b,[0 1])
13     c = a;
14     return;
15 end
16 a_1 = a(1);
17 a_2 = a(2);
18 b_1 = b(1);
```

```

19 b_2 = b(2);
20 and11 = and(a_1, b_1);
21 and12 = and(a_1, b_2);
22 and21 = and(a_2, b_1);
23 xor1 = xor(and12, and21);
24 and22 = and(a_2, b_2);
25 c(1) = xor(xor1, and11);
26 c(2) = xor(and11, and22);
27 end

```

Once a β element has been found it is used to build the mapping matrix.

```

1 function [ T ] =
2 buildIsomorphicMapping222( beta, L1, L2, P1, P2 )
3 %Build an isomorphic mapping matrix
4 %using a beta that satisfies the
5 %requirement  $R(\beta) = 0$ 
6 T = [ 0 0 0 0 0 0 0 0 1 ]';
7 for i = 1:7
8     buf = powgf2222(beta, i, L1, L2, P1, P2);
9     T = [buf' T];
10 end
11 end

```

Appendix B: Estimating the PMF for Mutual Information

The original PMF of noisy leakage needs to be estimated to perform mutual information analysis. The analysis in this paper uses the histogram estimation in [5], first detailed in [28]. Matlab code to perform the estimation is given below.

```
1 function [ histogram ] = estimatePMF( leak )
2 %Reorganizes a noisy leakage into histogram bins.
3 N = numel(leak);
4 %Bin size
5 h = (3.49*std(leak)*N^(-1/3));
6 %Minimum of the leak
7 minL = min(leak);
8 %Group values into bins
9 histogram = floor((leak-minL)./h);
10 end
```

Appendix C: Attack Simulation Code

The following code performs a second-order attack on simulated share leakages. First a function is given to perform a mapping of an element from the AES finite field to the chosen construction.

```
1 function [ c ] = map( a,T )
2 if numel(a) == 8
3     c = mod(T*a',2)';
4 elseif numel(a) == 1
5     aBin = d2b(a,8);
6     cBin = mod(T*aBin',2)';
7     c = b2d(cBin);
8 end
9 end
```

Another two functions are given which quickly calculated mutual information between two data-sets and multivariate mutual information between three data-sets.

```
1 function [ I ] = IXY( X,Y )
2 %I = HX(X) - HXGY(X,Y);
3 %This was found to be the fastest way of doing this
4 I = 0;
5 N = numel(X);
6 iu = unique(X);
7 ju = unique(Y);
8 is = max(X);
9 js = max(Y);
10 x = zeros(1,is+1);
11 y = zeros(1,js+1);
12 xy = zeros(is+1,js+1);
13 for i = 1:N
```

```

14     xcurr = X(i);
15     ycurr = Y(i);
16     x(xcurr+1) = x(xcurr+1) + 1;
17     y(ycurr+1) = y(ycurr+1) + 1;
18     xy(xcurr+1,ycurr+1) = xy(xcurr+1,ycurr+1) + 1;
19 end
20 for i = iu
21     for j = ju
22         if xy(i+1,j+1) == 0
23             continue
24         end
25         pxy = (xy(i+1,j+1))/N;
26         px = (x(i+1))/N;
27         py = (y(j+1))/N;
28         I = I + pxy*log2((pxy)/(px*py));
29     end
30 end
31 end

```

```

1 function [ I ] = IXYZ(X,Y,Z )
2 %I = IXY(X,Y) - (HXY(X,Y)+HX(Y)-HXYZ(X,Y,Z)) ;
3 I = 0;
4 N = numel(X);
5 iu = unique(X);
6 ju = unique(Y);
7 ku = unique(Z);
8 is = max(X);
9 js = max(Y);
10 ks = max(Z);
11 x = zeros(1,is+1);
12 y = zeros(1,js+1);
13 z = zeros(1,ks+1);
14 xy = zeros(is+1,js+1);
15 xz = zeros(is+1,ks+1);
16 yz = zeros(js+1,ks+1);
17 xyz = zeros(is+1,js+1,ks+1);
18 for i = 1:N
19     xcurr = X(i);
20     ycurr = Y(i);
21     zcurr = Z(i);
22     x(xcurr+1) = x(xcurr+1) + 1;
23     y(ycurr+1) = y(ycurr+1) + 1;
24     z(zcurr+1) = z(zcurr+1) + 1;

```



```

25     xy(xcurr+1,ycurr+1) = xy(xcurr+1,ycurr+1) + 1;
26     xz(xcurr+1,zcurr+1) = xz(xcurr+1,zcurr+1) + 1;
27     yz(ycurr+1,zcurr+1) = yz(ycurr+1,zcurr+1) + 1;
28     xyz(xcurr+1,ycurr+1,zcurr+1) = xyz(xcurr+1,ycurr+1,zcurr
        +1) + 1;
29 end
30 for i = iu
31     for j = ju
32         for k = ku
33             if xyz(i+1,j+1,k+1) == 0
34                 continue
35             end
36             pxyz = (xyz(i+1,j+1,k+1))/N;
37             pxy = (xy(i+1,j+1))/N;
38             pxz = (xz(i+1,k+1))/N;
39             pyz = (yz(j+1,k+1))/N;
40             px = (x(i+1))/N;
41             py = (y(j+1))/N;
42             pz = (z(k+1))/N;
43             I = I + pxyz*log2((pxy*pxz*pyz)/(pxyz*px*py*pz));
44         end
45     end
46 end
47 end

```

The actual attack is given below.

```

1  %HW ATTACK
2  clc;clear;close all;
3  load T_array.mat
4  num_correct = 0;
5  %Number of samples
6  N = 80000;
7  I = eye(8,8);
8  %Set 2 Good Set
9  T_temp_1 = cell2mat(T_array(1));
10 T_temp_2 = cell2mat(T_array(25));
11 T1Set2 = T_temp_1(:, :, 1);
12 T2Set2 = T_temp_2(:, :, 5);
13 %Set 1 Bad Set
14 T_temp_1 = cell2mat(T_array(1));
15 T_temp_2 = cell2mat(T_array(1));
16 T1Set1 = T_temp_1(:, :, 1);

```

```

17 T2Set1 = T_temp_2(:, :, 2);
18 %Sbox Invert Lookup Table
19 SBINV = [{ '52' } '09' '6a' 'd5' '30' '36' ...
20 'a5' '38' 'bf' '40' 'a3' '9e' '81' 'f3' 'd7' 'fb' ...
21 '7c' 'e3' '39' '82' '9b' '2f' 'ff' '87' '34' '8e' ...
22 '43' '44' 'c4' 'de' 'e9' 'cb' ...
23 '54' '7b' '94' '32' 'a6' 'c2' '23' '3d' 'ee' ...
24 '4c' '95' '0b' '42' 'fa' 'c3' '4e' ...
25 '08' '2e' 'a1' '66' '28' 'd9' '24' 'b2' ...
26 '76' '5b' 'a2' '49' '6d' '8b' 'd1' '25' ...
27 '72' 'f8' 'f6' '64' '86' '68' '98' '16' ...
28 'd4' 'a4' '5c' 'cc' '5d' '65' 'b6' '92' ...
29 '6c' '70' '48' '50' 'fd' 'ed' 'b9' 'da' ...
30 '5e' '15' '46' '57' 'a7' '8d' '9d' '84' ...
31 '90' 'd8' 'ab' '00' '8c' 'bc' 'd3' '0a' ...
32 'f7' 'e4' '58' '05' 'b8' 'b3' '45' '06' ...
33 'd0' '2c' '1e' '8f' 'ca' '3f' '0f' '02' ...
34 'c1' 'af' 'bd' '03' '01' '13' '8a' '6b' ...
35 '3a' '91' '11' '41' '4f' '67' 'dc' 'ea' ...
36 '97' 'f2' 'cf' 'ce' 'f0' 'b4' 'e6' '73' ...
37 '96' 'ac' '74' '22' 'e7' 'ad' '35' '85' ...
38 'e2' 'f9' '37' 'e8' '1c' '75' 'df' '6e' ...
39 '47' 'f1' '1a' '71' '1d' '29' 'c5' '89' ...
40 '6f' 'b7' '62' '0e' 'aa' '18' 'be' '1b' ...
41 'fc' '56' '3e' '4b' 'c6' 'd2' '79' '20' ...
42 '9a' 'db' 'c0' 'fe' '78' 'cd' '5a' 'f4' ...
43 '1f' 'dd' 'a8' '33' '88' '07' 'c7' '31' ...
44 'b1' '12' '10' '59' '27' '80' 'ec' '5f' ...
45 '60' '51' '7f' 'a9' '19' 'b5' '4a' '0d' ...
46 '2d' 'e5' '7a' '9f' '93' 'c9' '9c' 'ef' ...
47 'a0' 'e0' '3b' '4d' 'ae' '2a' 'f5' 'b0' ...
48 'c8' 'eb' 'bb' '3c' '83' '53' '99' '61' ...
49 '17' '2b' '04' '7e' 'ba' '77' 'd6' '26' ...
50 'e1' '69' '14' '63' '55' '21' '0c' '7d' ];
51 for i = 1:256
52     sBoxInvDec(i) = hex2dec(cell2mat(SBINV(i)));
53 end
54 for guess = 0:255
55     for outi = 0 :255
56         Z_look_up(guess+1,outi+1) =...
57             sum(map(d2b(sBoxInvDec(...
58                 bitxor(outi, guess)+1),8),T2Set2));
59     end

```

```

60 end
61 %100 sets
62 for attempt = 1:100
63     %Choose a random secret
64     secret = floor(rand*256);
65     %Generate a set of outputs
66     out = floor(rand(1,N)*256);
67     %Generate mask values, share b0
68     b0 = floor(rand(2,N)*256);
69     for i = 1:N
70         %b1 is equal to calculated output
71         %with a leakage function
72         b1(i) = bitxor(...
73             sBoxInvDec(bitxor(out(i),secret)+1),b0(1,i));
74         %Map both leakages to the different constructions.
75         leak1(i) = sum(map(d2b(b0(1,i),8),T1Set2));
76         leak2(i) = sum(map(d2b(b1(i),8),T2Set2));
77         perc = i/N;
78     end
79
80     %Noise Level in Gaussian STD
81     s = 3;
82     noise1 = s*randn(1,N);
83     noise2 = s*randn(1,N);
84     SNR1 = snr(leak1,noise1);
85     SNR2 = snr(leak2,noise2);
86     leak1 = leak1+noise1;
87     leak2 = leak2+noise2;
88
89     %Product Combining Function
90     product_combining = (leak1 - mean(leak1)) .* ...
91         (leak2 - mean(leak2));%abs(leak1-leak2);%
92     %With noise need to estimate pmf for MMIA
93     if s > 0
94         pmf1 = estimatePMF(leak1);
95         pmf2 = estimatePMF(leak2);
96     %No noise just use possible values of the HW function as
97         histogram bins
98     else
99         pmf1 = leak1;
100         pmf2 = leak2;
101     end

```

```

102     for guess = 0:255
103         for i = 1:N
104             intermediate(i) = Z_look-up(guess+1,out(i)+1);
105         end
106         %Uncomment phi for MMIA, corr for HO-DPA
107         corr(guess+1) = ...
108             correlation(intermediate,product_combining);
109         %phi(guess+1) = IXYZ(pmf1,pmf2,intermediate);
110     end
111     %Corr for HO-DPA, phi for MMIA
112     if min(corr) == corr(secret+1)
113         num_correct = num_correct + 1;
114     end
115     %if min(phi) == phi(secret+1)
116     %     num_correct = num_correct +1;
117     %end
118
119     %More than 10 of the 100 failed
120     if attempt > num_correct+9
121         break
122     end
123     %Got the first ten correct, so assume the rest will
124     %follow suit
125     if attempt == 10 && num_correct == 10
126         break
127     end
128     end
129     %This block for HO-DPA
130     figure
131     hold on
132     plot(corr)
133     plot(secret+1,corr(secret+1),'o');
134     title('Ho-DPA');
135
136     %This block for mmia
137     % figure
138     % hold on
139     % plot(phi);
140     % plot(secret+1,phi(secret+1),'o');
141     % title('MMIA');

```

Bibliography

- [1] X. Zhang and K. Parhi, “High-speed VLSI architectures for the AES algorithm,” *IEEE Trans. on VLSI Systems*, vol. 12, no. 9, pp. 957–967, 2004.
- [2] L. De Meyer, O. Reparaz, and B. Bilgin, “Multiplicative masking for aes in hardware,” *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 431–468, 2018.
- [3] P. Kocher, J. Jaffe, and B. Jun, “Differential Power Analysis,” in *Annual Intl. Cryptology Conf.*, pp. 388–397, Springer, 1999.
- [4] E. Brier, C. Clavier, and F. Olivier, “Correlation power analysis with a leakage model,” in *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 16–29, Springer, 2004.
- [5] N. Veyrat-Charvillon and F.-X. Standaert, “Mutual information analysis: how, when and why?,” in *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 429–443, Springer, 2009.
- [6] S. Chari, J. R. Rao, and P. Rohatgi, “Template attacks,” in *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 13–28, Springer, 2002.
- [7] E. Prouff, M. Rivain, and R. Bevan, “Statistical analysis of second order differential power analysis,” *IEEE Trans. on computers*, vol. 58, no. 6, pp. 799–811, 2009.
- [8] A. Duc, S. Dziembowski, and S. Faust, “Unifying leakage models: From probing attacks to noisy leakage,” in *Annual Intl. Conf. on the Theory and Applications of Cryptographic Techniques*, pp. 423–440, Springer, 2014.
- [9] S. Mangard, N. Pramstaller, and E. Oswald, “Successfully attacking masked aes hardware implementations,” in *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 157–171, Springer, 2005.
- [10] S. Mangard and K. Schramm, “Pinpointing the side-channel leakage of masked aes hardware implementations,” in *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 76–90, Springer, 2006.

- [11] X. Zhang and Y. Lao, “On the construction of composite finite fields for hardware obfuscation,” *IEEE Trans. Computers*, 2019.
- [12] X. Zhang, *VLSI architectures for modern error-correcting codes*. Crc Press, 2015.
- [13] Pub, NIST FIPS, “197: Advanced encryption standard (AES),” *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.
- [14] D. Canright, “A very compact s-box for aes,” in *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 441–455, Springer, 2005.
- [15] C. Paar, *Efficient architectures for computation in Galois fields*. PhD thesis, Ruhr-Universitat Bochum, 1994.
- [16] T. De Cnudde *et al.*, “Masking aes with $d + 1$ shares in hardware,” in *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 194–212, Springer, 2016.
- [17] H. Groß, S. Mangard, and T. Korak, “Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order,” in *TIS@CCS*, p. 3, 2016.
- [18] M.-L. Akkar and C. Giraud, “An implementation of DES and AES, secure against some attacks,” in *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 309–318, Springer, 2001.
- [19] L. Genelle, E. Prouff, and M. Quisquater, “Thwarting higher-order side channel analysis with additive and multiplicative maskings,” in *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 240–255, Springer, 2011.
- [20] M. Nassar *et al.*, “Rsm: a small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs,” in *Design, Automation & Test in Europe Conference & Exhibition*, pp. 1173–1178, IEEE, 2012.
- [21] S. Nikova, C. Rechberger, and V. Rijmen, “Threshold implementations against side-channel attacks and glitches,” in *Intl. Conf. on Information and Communications Security*, pp. 529–545, Springer, 2006.
- [22] B. Gierlichs *et al.*, “Mutual Information Analysis,” in *Conf. on Cryptographic Hardware and Embedded Systems*, pp. 426–442, Springer, 2008.
- [23] S. Chari *et al.*, “Towards sound approaches to counteract power-analysis attacks,” in *Annual Intl. Cryptology Conf.*, pp. 398–412, Springer, 1999.
- [24] B. Gierlichs *et al.*, “Revisiting higher-order dpa attacks,” in *Cryptographers Track at the RSA Conf.*, pp. 221–234, Springer, 2010.

- [25] G. Becker, J. Cooper, E. DeMulder, G. Goodwill, J. Jaffe, G. Kenworthy, T. Kouzminov, A. Leiserson, M. Marson, P. Rohatgi, *et al.*, “Test vector leakage assessment (tvla) methodology in practice,” in *Intl. Cryptographic Module Conf.*, vol. 1001, p. 13, 2013.
- [26] O. Reparaz, “Detecting flawed masking schemes with leakage detection tests,” in *Intl. Conf. on Fast Software Encryption*, pp. 204–222, Springer, 2016.
- [27] G. Fumaro, A. Martinelli, E. Prouff, and M. Rivain, “Affine masking against higher-order side channel analysis,” in *Intl. Workshop on Selected Areas in Cryptography*, pp. 262–280, Springer, 2010.
- [28] D. W. Scott, “On optimal and data-based histograms,” *Biometrika*, vol. 66, no. 3, pp. 605–610, 1979.