8-2017

# Reusable Garbled Circuit Implementation of AES to Avoid Power Analysis Attacks

Venkata Lakshmi Sudheera Bommakanti
*Clemson University*

# Reusable Garbled Circuit Implementation of AES to Avoid Power Analysis Attacks

---

A Thesis
Presented to
the Graduate School of
Clemson University

---

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Venkata Lakshmi Sudheera Bommakanti
August 2017

---

Accepted by:
Dr. Richard Brooks, Committee Chair
Dr. Richard Groff
Dr. Rajendra Singh

# Abstract

Unintended side-channel leaks can be exploited by attackers and achieved quickly, and using relatively inexpensive equipment. Cloud providers aren’t equipped to provide assurances of security against such attacks. One most well-known and effective of the side-channel attack is on information leaked through power consumption. Differential Power Analysis (DPA) can extract a secret key by measuring the power used while a device is executing the any algorithm. This research explores the susceptibility of current implementations of Circuit Garbling to power analysis attacks and a simple variant to obfuscate functionality and randomize the power consumption reusing the garbling keys and the garbled gates. AES has been chosen as an example. The first task is to implement the garbled variants of basic logic gates in hardware (RTL design) using Circuit Garbling. The second task is to use the above created gates and create an RTL implementation of AES using Verilog HDL. The next task is to perform a Differential Power Analysis(DPA) on this circuit and evaluate its resilience to attack.

# Dedication

I dedicate this thesis to my parents Ramakrishna and Meenakshi, my brother Yashwanth and my advisor Dr. Richard Brooks for their patience and encouragement.

# Acknowledgements

I would like to thank my advisor Dr. Richard Brooks for believing in my abilities, providing me with an opportunity to pursue this research and providing invaluable guidance and support. Working with him and his team has been a fantastic learning experience. I thank Dr. Rajendra Singh for his encouragement and valuable suggestions towards taking up this project and being a part of my committee. I would like to thank Dr. Richard Groff for being a part of my committee and reviewing my thesis. I extend my gratitude towards Dr. Yingjie Lao and senior PhD candidate Nilim Sarma for helping me solve design challenges and providing me with resources to understand underlying concepts. I am also grateful to Dr. Brook's research team for their amicable support. Finally, I appreciate Clemson University for their wonderful research facilities.

# Table of Contents

Page

# List of Figures

Page

# Chapter 1

# Introduction

Power analysis is a form of side channel attack in which the attacker studies the power consumption of a cryptographic hardware device (such as a smart card, tamper-resistant "black box", or integrated circuit). The attack has been proven to extract cryptographic keys and other secret information non-invasively from the device. If the device is in the attacker's possession, the attacker may exploit physical side-channels such as power consumption [1], emitted radiation [2], timing, vibration, and sometimes the memory cache [3]. Even when there is no computation being performed, cold boot attacks may be performed [4]. Hence, security requires more than algorithmic soundness. Circuit garbling has been proposed to render hardware leakage resilient; however, until recent years it has been viewed as being of limited practical significance [5] due to the size of hardware produced. This research focuses on hardware reusability and an intuitive method of reusing garbling keys to increase the resistance to reverse engineering and side-channel analysis.

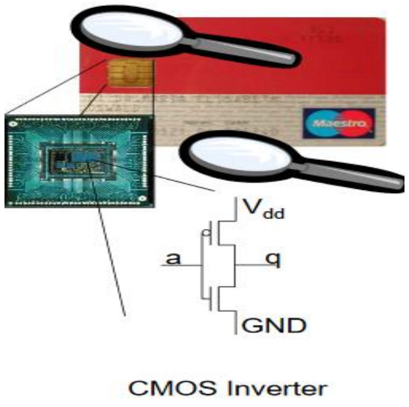


Figure 1.1: Power analysis attacks. [6]

## 1.1 Secure Function Evaluation

Efficient Secure Function Evaluation (SFE) in a significantly untrusted environment is a longstanding goal for modern security. The goal of two-party SFE is to let two (polynomially bounded) parties that don’t trust each other compute an arbitrary function on their private inputs without revealing information about the inputs, beyond the output of the function. SFE has a variety of applications, particularly in settings with strong security and privacy demands. Deployment of SFE has been very limited and believed to be expensive until recent improvements in algorithms, code generation, computing platforms and networks. [6]

Example of SFE: The Millionaires’ problem.

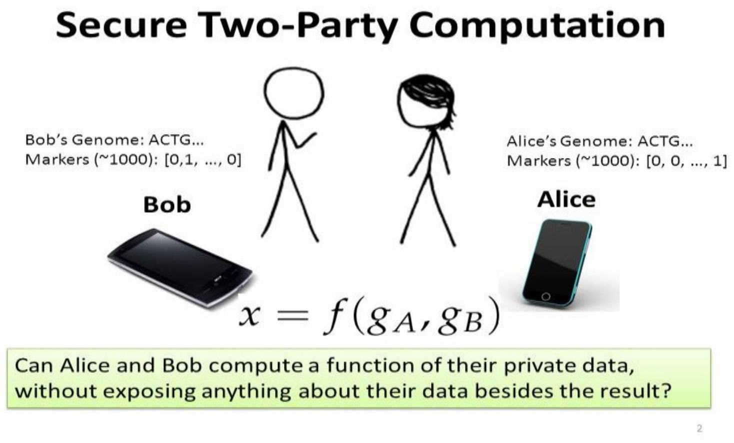

Secure Two-Party Computation
Bob's Genome: ACTG...
Markers (~1000): [0,1, ..., 0]
Bob
Alice's Genome: ACTG...
Markers (~1000): [0, 0, ..., 1]
Alice
$x = f(g_A, g_B)$
Can Alice and Bob compute a function of their private data, without exposing anything about their data besides the result?
2

Figure 1.2: Secure Function Evaluation illustration [5]

## 1.2 Preliminaries on Garbled Circuit

Let's discuss the circuit garbling algorithm in this section, Yao's Garbled Circuit (GC) enables two parties, a sender S with private input y and receiver a R with private input x, to securely compute a Boolean circuit C on (x, y) without revealing any information other than the result z = C (x, y) of the computation. Even the intermediate values are not revealed.

### 1.2.1 Garbled Circuit Protocol

The sender is the circuit constructor S and creates a garbled circuit $G_i G_i$ from the circuit C: for each wire $w_i$ $w_i$ of C, two garblings: $w_i^0$ , $w_i^1$ where $w_i^j$ is the garbled value $w_i$ $w_i$'s value j are chosen randomly. Also, for each gate $G_i$, S creates a garbled table $T_i$ with the following property: given a set of garbled values of $G_i G_i$'s inputs, $T_i$ allows to recover the garbled value of the corresponding $T_i$'s output, but nothing else. Sender S sends these garbled tables, called garbled circuit $G_i$GC to receiver R. Hence unless and until S informs him, R does not know the functionality of $G_i$.

Then, R gets the garbled inputs $w_i w_i$ corresponding to the inputs of both parties: the garbled inputs corresponding to the inputs y of S are sent directly$\tilde{y}_i = \tilde{y}_i^{y_k}\tilde{y}_i = \tilde{y}_i^{y_k}$. For each of R's inputs$x_i$, both parties run a 1-out-of-2 Oblivious Transfer (OT) protocol, where S inputs y and R inputs $x_i$. $x_i$The OT protocol ensures that R receives only the garbled value corresponding to his input bit while S learns nothing about$x_i$. $x_i$Now, R evaluates the garbled circuit GC on the garbled inputs to obtain the garbled output z by evaluating GC gate by gate, using the garbled table$T_i$ . $T_i$Finally, R determines the plain value z corresponding to the obtained garbled output value using an output translation table sent by S. [7]

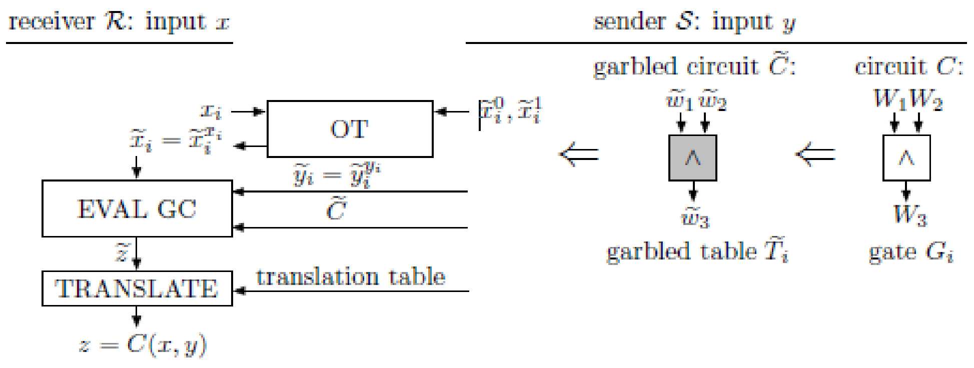

receiver $\mathcal{R}$: input $x$
sender $\mathcal{S}$: input $y$
garbled circuit $\widetilde{C}$:
circuit $C$:
$x_i$
$\widetilde{x}_i = \widetilde{x}_i^{x_i}$
OT
$\widetilde{x}_i^0, \widetilde{x}_i^1$
$\widetilde{w}_1 \widetilde{w}_2$
$W_1 W_2$
$\wedge$
$\wedge$
$\widetilde{y}_i = \widetilde{y}_i^{y_i}$
EVAL GC
$\widetilde{C}$
$\widetilde{w}_3$
$W_3$
garbled table $\widetilde{T}_i$
gate $G_i$
$\widetilde{z}$
TRANSLATE
translation table
$z = C(x, y)$

Figure 1.3: Yao’s garbled circuit for AND gate. [7]

### 1.2.2 Algorithm Construction

- Randomly choose a global key $R \in \{0,1\}^N$.
- For each input wire $W_i$ of C,

Randomly choose its garbled value $w_i^0 = \langle k_i^0, p_i^0 \rangle \in \{0,1\}^{N+1}$

Set the other garbled value $w_i^1 = \langle k_i^1, p_i^1 \rangle = \langle k_i^0 xorR, p_i^0 xor1 \rangle$

- For each gate $G_i$ of C in topological order

  Label it with index i.

  If $G_i$is a 2-input gate $W_c = g\langle W_a, W_b \rangle$ with garbled input values $w_a^0 = \langle k_a^0, p_a^0 \rangle, w_a^1 = \langle k_a^1, p_a^1 \rangle, w_b^0 = \langle k_b^0, p_b^0 \rangle, w_b^1 = \langle k_b^1, p_b^1 \rangle$ :

  - Randomly choose the garbled value $w_c^0 = \langle k_c^0, p_c^0 \rangle \in \{0,1\}^{N+1}$
  - Randomly choose garbled output $w_c^1 = \langle k_c^1, p_c^1 \rangle = \langle k_c^0 xorR, p_c^0 xor1 \rangle$
  - Create garbled table for each of 2^2 possible combinations of $G_i's$ input values $v_a, v_b \in \{0,1\}$, set

$$e_{v_a, v_b} = H\left(k_a^{v_a} | k_b^{v_b} | i\right) xor w_c^{g_i(v_a, v_b)}$$

$$sortentries \in thetablebyinputpointers$$

- for each circuit-output wire $W_i$ of the gate $G_j$ with garblings $w_i^0 = \langle k_i^0, p_i^0 \rangle$and $w_i^1 = \langle k_i^1, p_i^1 \rangle$:

  create garbled output table for both the possible output values $v \in (0,1)$.

  Set $e_v = H(k_i^v | out | j) xor v$

  sort entries e in the table by the input pointers, i.e., place entry $e_v$in position $p_i^v$.

### 1.2.3 Algorithm Evaluation

for each circuit wire $W_i$of C

receive corresponding garbled value $w_i = \langle k_i, p_i \rangle$

- for each gate $G_i$

  $G_i$is a 2-input gate $W_c = g\langle W_a, W_b \rangle$ with garbled input values $w_a^0 = \langle k_a^0, p_a^0 \rangle, w_a^1 = \langle k_a^1, p_a^1 \rangle, w_b^0 = \langle k_b^0, p_b^0 \rangle, w_b^1 = \langle k_b^1, p_b^1 \rangle$:

  - Decrypt garbled values from garbled table entry e in position $\langle p_a, p_b \rangle : w_c = \langle k_c, p_c \rangle = H(k_a^{v_a} | k_b^{v_b} | i) xor e.$

  For each circuit's output wire $W_i$(output of gate $G_j$) with garbling $w_i = \langle k_i, p_i \rangle$

  - Decrypt the output value $f_i$ from garbled output table entry e in row $p_i : f_i = H(k_i | out | j) xor e.$ [8]

  -

### 1.2.4 GC Example

Consider an example of evaluating a black box device made of AND gate. The device has been fabricated as a garbled AND gate: the functionality remains the same but the implementation differs based on garbling keys. In this case, the manufacturer is the sender S and the customer is the receiver R. S creates garbling keys and garbling table T for the gate and the garbling key for its input.

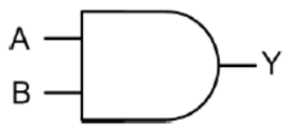


Figure 1.4: Original AND gate

| **Input values** | **Encoding** | **output** |
|---|---|---|
| A=0, B=0 | E(ka0, kb0) | ky0 |
| A=0, B=1 | E(ka0, kb1) | ky0 |
| A=1, B=0 | E(ka1, kb0) | ky0 |
| A=1, B=1 | E(ka1, kb1) | ky1 |

Table 1.1: Garbling table T sent by S to R

Where ka0, ka1, kb0, kb1 are the keys for input wires A and B and ky0, ky1 are the keys for the output wires. R is given keys corresponding to his inputs and S's garbled input: either ka0 or ka1. R then feeds them to the table and will be able to decode the row corresponding to the input keys he has. S can either embed a de-garbling function at the end for R to understand the outputs or may inform R about the interpretation of the output.

The following observations should be noted from the above example:

- R has no access to S' inputs or garbling keys.
- Functionally, the black box device is an AND gate, but the presence of garbling keys and encoding E of the keys obfuscates the functionality. Hence it makes reverse engineering a bigger challenge.
- Randomized power consumption: changing garbling keys would change the encoding, contents of the table and hence power consumption of the device. This property is explored further in Chapters 3 and 4 of this thesis.

- R can be an adversary or a vulnerable user. Since R has no access to garbling keys and he isn't aware of the algorithm used by the device to generate the outputs, he cannot tamper the device.

In this research, AES key and the input plain text are both considered to be R's inputs. The inputs are garbled in the circuit after they are fed by the user. The attacker is assumed to have access to the device and to the power consumed by the device.

## 1.3 Side Channel attacks: Power Analysis Attacks

In a typical cryptosystem utilizing a block cipher, the cryptanalysis will typically start with a focus on the primary inputs and outputs as potential sources of information for an attack. These include the plaintext input, the secret key, and the resulting ciphertext output. When the device is used in real world, side channels come in to existence and they might leak information about the device's operation states or might also correlate with the secret key which can be exploited by an attack. The following figure illustrates the typical side channels in a system. [9]

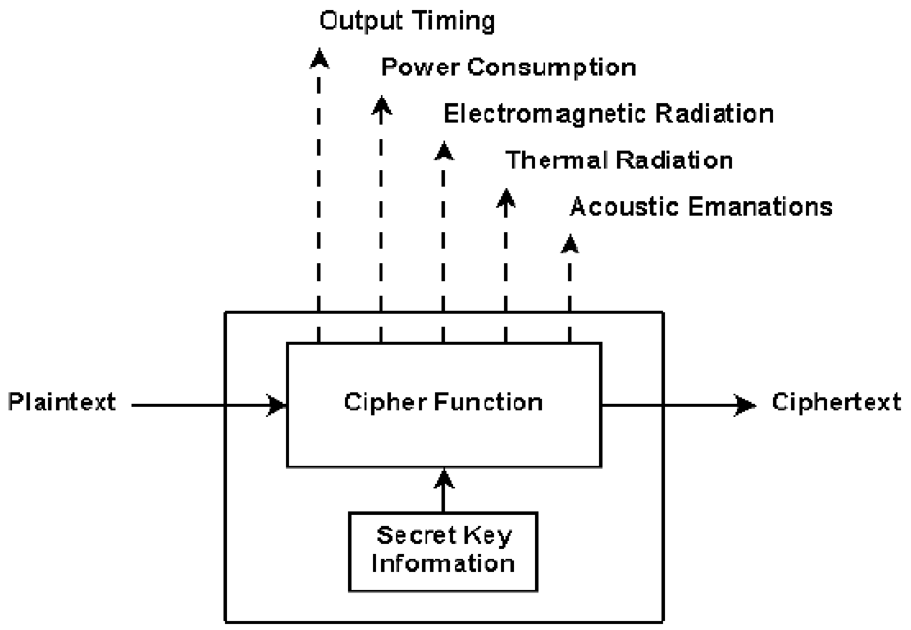


Figure 1.5: Side channels in a system [9]

Power analysis attack are the most common and power side-channel analysis attack that utilizes information leaked by the power supply of a system during its operation. These attacks intend to find a correlation between the instantaneous power consumption and the internal state of a system.

It can be decomposed into the following steps:

1. Identify and establish a relationship between secret key and the instantaneous power consumption. Also need to determine the required inputs to the system, the output values to be measured, and when to capture them.

2. Try to Extract the state of the system by measuring and recording the items identified in the previous step including the power consumption the measurements called the traces can be collected in a non-invasive manner when the operation is performed by the device. [10]

3. Evaluate the relationship between the measured items by processing the extracted information.

At an algorithm level, execution of different operations consumes different amounts of power. This difference is generally independent of the data being manipulated. Hence, by examining the power trace, it might be possible to infer what operations are being performed. For instance, Figure below shows the instantaneous power consumption of a device while it performs the Data Encryption Standard (DES) operation. The 16 individual rounds of DES are clearly visible from the repetitive patterns. However, the individual data bits being manipulated in the cipher cannot be visually determined.

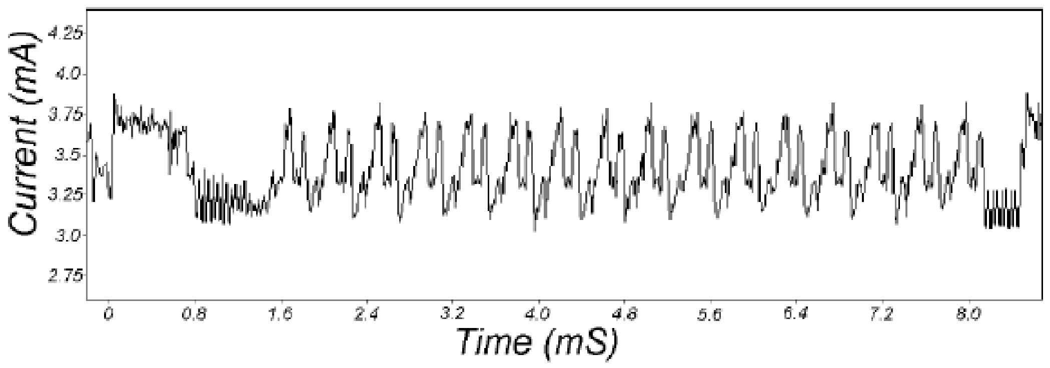


Figure 1.6: Power Tarce of DES[1]

Despite not being able to directly ascertain data values, this form of analysis can be still be useful. It can be used to setup more powerful attacks by identifying the point within a cryptographic operation at which to attack.

Differences in instantaneous power consumption can be related to the bits that are being manipulated as well. When the bit values change the hardware consumes different amounts of power but at a lower scale [12]. These variations can be monitored and the bit values can hence be determined and therefore, the secret keys in the ciphers get compromised. However, owing to the subtle the power variations, detection is more difficult. It would require modifications to the hardware and some statistical techniques to find and correlate the individual bit values. It should be noted that unlike physical attacks, power analysis attacks are non-invasive, easily-automated, and can be mounted without any prior knowledge of the design of the target device also, these attacks cannot generally be detected by a device since the adversary can also be passive.

### 1.3.1 Simple Power Analysis (SPA)

The most basic form of power analysis is the Simple Power Analysis (SPA). This consists of a process of examining power traces for large scale differences which are caused by the operations being performed.

It has been shown that in some applications, it is possible to determine which software instructions are occurring or possibly which data bits are being changed [1]. In a worst case, situation, it may be even be possible to determine the data bit values of secret information. Implementations where these differences are dependent on data values being manipulated are more vulnerable to an attack. This attack is very effective when an attacker has unlimited access to perform many encryption or decryption operations a device and also when the algorithm is known.

### 1.3.2 Differential Power Analysis (DPA)

Our computers and microchips on smart cards leak information about their functionality while they execute their processes. Modern cryptographic devices are realized in hardware using semiconductor logic gates, which are made from transistors. When charge is applied to or removed from transistor's gate, electrons flow in the silicon substrate consuming power and also produced electromagnetic radiation. To measure a circuit's power consumption, a small resistor is inserted in series with the power or ground input. The voltage difference across the resistor divided by the resistance yields the current which depends on the operation being performed and the bit values being manipulated. [9]

In addition to large-scale power variations due to the instruction sequence, there are effects correlated to data values being manipulated. These variations tend to be smaller and are sometimes overshadowed by measurement errors and other noise. In such cases, it is still often possible to break the system using statistical functions tailored to the target algorithm. [10]The DPA selection function D(C, b, Ks) is the value of an AES intermediate state which is noted before it is fed to the next state for ciphertext C, K is the AES key used for that computation and b is the bit being used for separating sets. The attacker then observes m operations and captures $T_{1:m}$[1:k] and also notes down the cipher texts $C_{1:m}$.

Given the $T_{1:m}$[1:k] power traces and corresponding ciphertext values C1:m two groups of traces are constructed: one where D(C, b, Ks) = 0 and another where D(C, b, Ks) = 1. Next the attacker computes a *k*-sample differential trace ΔD[j] by finding the difference between the average of traces for which D(C, b, Ks) = 1 and the average of traces for which D(C, b, Ks) = 0. Therefore ΔD[j] is the average over $C_{1:m}$of the effect due to the value represented by the selection function on the power consumption measurements at point $j$. The DPA algorithm is shown in in the below equation.[9]

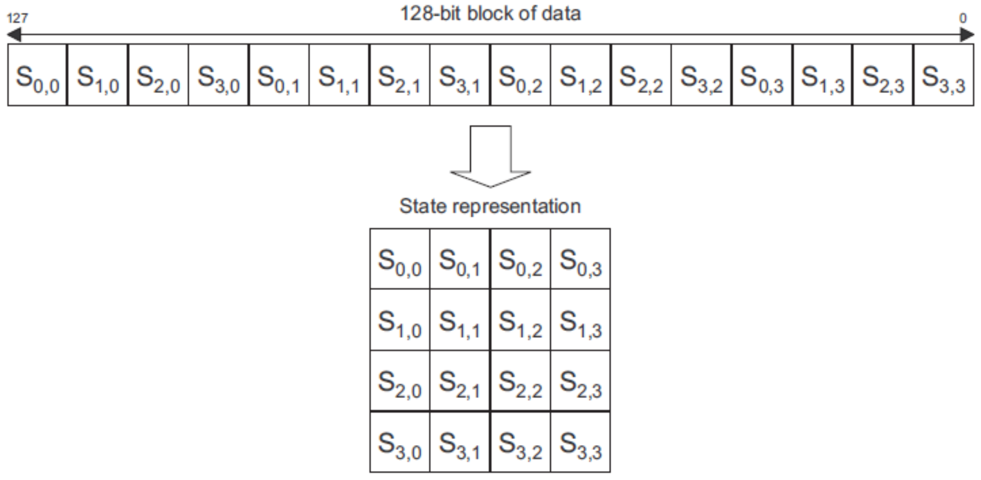
$$\begin{aligned}\Delta_D[j] &= \frac{\sum_{i=1}^{m} D(C_i, b, K_s)\mathbf{T}_i[j]}{\sum_{i=1}^{m} D(C_i, b, K_s)} - \frac{\sum_{i=1}^{m} (1 - D(C_i, b, K_s))\,\mathbf{T}_i[j]}{\sum_{i=1}^{m} (1 - D(C_i, b, K_s))} \\ &\approx 2\left(\frac{\sum_{i=1}^{m} D(C_i, b, K_s)\mathbf{T}_i[j]}{\sum_{i=1}^{m} D(C_i, b, K_s)} - \frac{\sum_{i=1}^{m} \mathbf{T}_i[j]}{m}\right)\end{aligned}$$

If the key guess *Ks* is correct, the average trace for D(C, b, Ks) = 1 will be slightly higher at the point of correlation and the average trace for D(C, b, Ks) = 0 will be slightly lower. Therefore the bit computed by the selection function D(C, b, Ks) will equal the actual value of target bit *b* with probability P = 1. It is effectively correlated in this case to what was computed by the target device. If the key guess *Ks* is incorrect, the bit computed by the selection function D(C, b, Ks) will equal the correct bit value with probability P = ½ or half the ciphertexts. It is effectively uncorrelated in this case to what was computed by the target device. [9]

Hence it is imperative from this section that the dependence of the device's power trace on the bit values being manipulated and the function(C) being implemented in the hardware. Therefore, to make the analysis complex, the circuit C can be garbled so that the function is now dependent on the garbling keys which can be changed using a simple hardware like LFSR at requisite periods. In the presence of garbling the power trace at any state for a given set of inputs to the garbled circuit GC of C would depend on the garbling keys and the bit values being manipulated in the current state hence if the garbling keys are manipulated even by a single bit the above analysis would fail. This has been proved in chapter 4 of this research.

## 1.4 Advanced Encryption Standard(AES)

AES is a symmetric-block-cipher algorithm that accepts 128-bit input plain text and outputs a 128-bit encrypted text. The data is arranged and processed in a 4x4 matrix each field of the matrix is comprised of 8 bits.



127
128-bit block of data
0
$S_{0,0}$ $S_{1,0}$ $S_{2,0}$ $S_{3,0}$ $S_{0,1}$ $S_{1,1}$ $S_{2,1}$ $S_{3,1}$ $S_{0,2}$ $S_{1,2}$ $S_{2,2}$ $S_{3,2}$ $S_{0,3}$ $S_{1,3}$ $S_{2,3}$ $S_{3,3}$
State representation
$S_{0,0}$ $S_{0,1}$ $S_{0,2}$ $S_{0,3}$
$S_{1,0}$ $S_{1,1}$ $S_{1,2}$ $S_{1,3}$
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{3,0}$ $S_{3,1}$ $S_{3,2}$ $S_{3,3}$

Figure 1.7: input 128-bit text to 4x4 state matrix [9]

The encryption process is broken in to 4 steps: sub-bytes, shift-rows, mix-columns and add-Round-key. Each round also needs a unique key generated by the key generation logic.

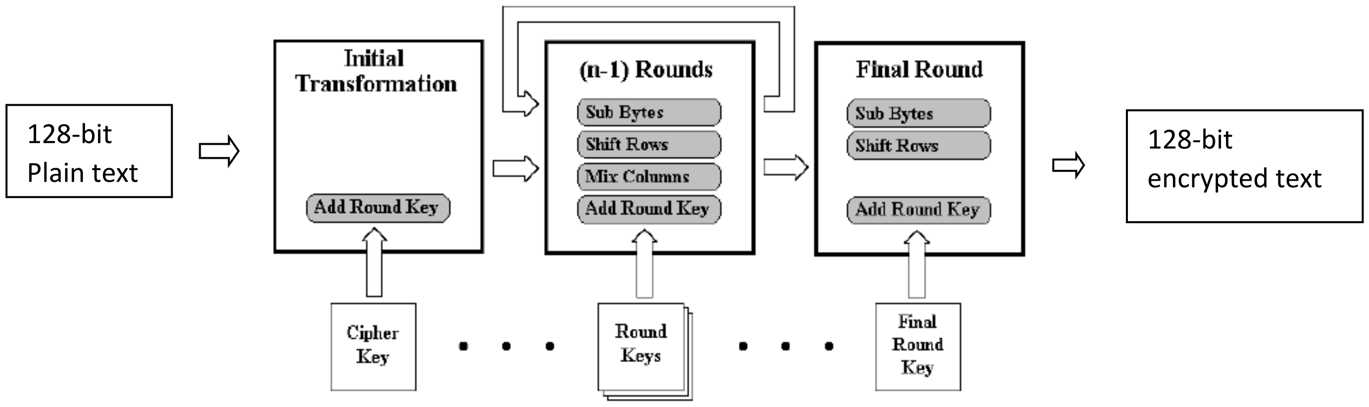
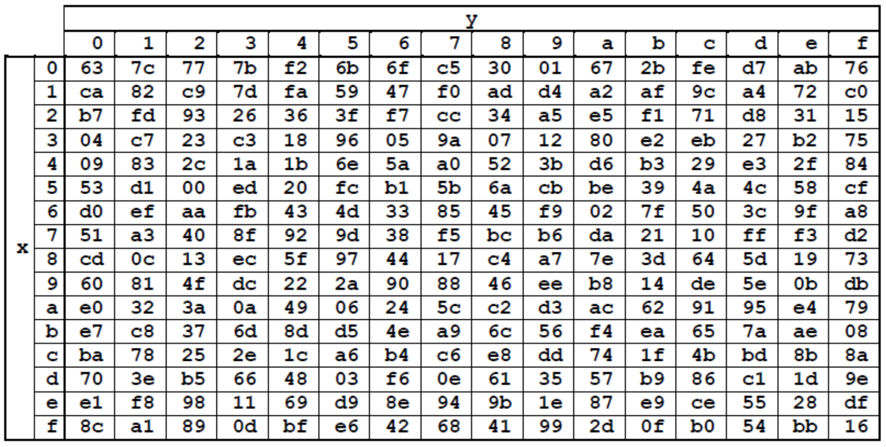


Figure 1.8: AES algorithm flow [13]

The arithmetic corresponding to various operations of the algorithm are performed in the finite field GF(2^8). Hence the addition and subtraction are performed modulo 2 [9].

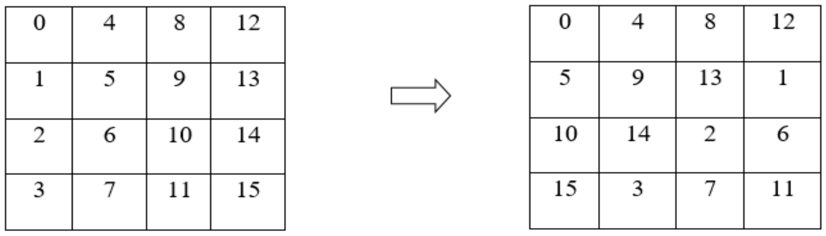
**Sub-bytes:** It’s a non-linear transformation in which each byte in the state matrix is replaced by the corresponding byte from an S-box table [9]. The S-box is as shown below:



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

Figure 1.9: AES S-box [14]

**Shift-rows:** This step shifts the rows of state to provide a horizontal diffusion. The first row is left as it is, second row is shifted left by one position, the third row is shifted left by two positions and the fourth row is shifted left by 3 positions.



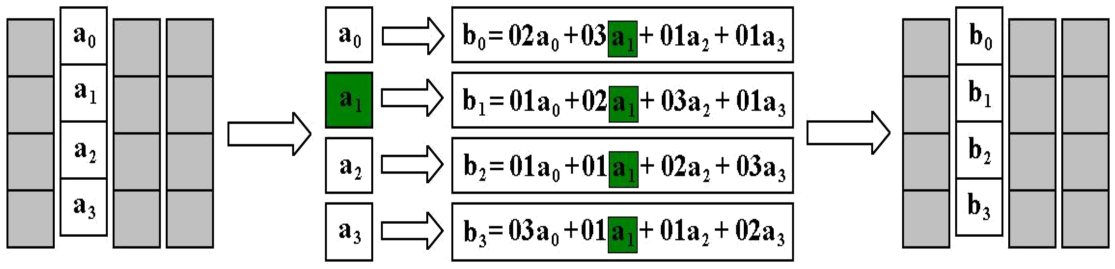
| | | | |
|---|---|---|---|
| 0 | 4 | 8 | 12 |
| 1 | 5 | 9 | 13 |
| 2 | 6 | 10 | 14 |
| 3 | 7 | 11 | 15 |

| | | | |
|---|---|---|---|
| 0 | 4 | 8 | 12 |
| 5 | 9 | 13 | 1 |
| 10 | 14 | 2 | 6 |
| 15 | 3 | 7 | 11 |

Figure 1.10: Shift-rows transformation [13]

**Mix-Columns:** The Mix-Columns transformation treats each word as a four term polynomial in a Galois Field GF(2^8) to provide vertical diffusion. Each polynomial is multiplied modulo x^4 + 1 with a fixed polynomial (3x^3+x^2+x+2). This is the most processing intensive transformation in AES since it involves a multiply operation. [9]

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$



$b_0 = 02a_0 + 03a_1 + 01a_2 + 01a_3$

$b_1 = 01a_0 + 02a_1 + 03a_2 + 01a_3$

$b_2 = 01a_0 + 01a_1 + 02a_2 + 03a_3$

$b_3 = 03a_0 + 01a_1 + 01a_2 + 02a_3$

Figure 1.11: Mix-Columns Step [13]

**Add-Round-Key:** It is a bit wise OR operation between the key generated for that round and the output of that round.

**Key generation schedule:** the input 128 bit cipher key w[0] to w[3] is used to generate the round key 1 which is used in the add round key operation and then used to generate round key 2. This process is continued till round key 10 is generated. [2]

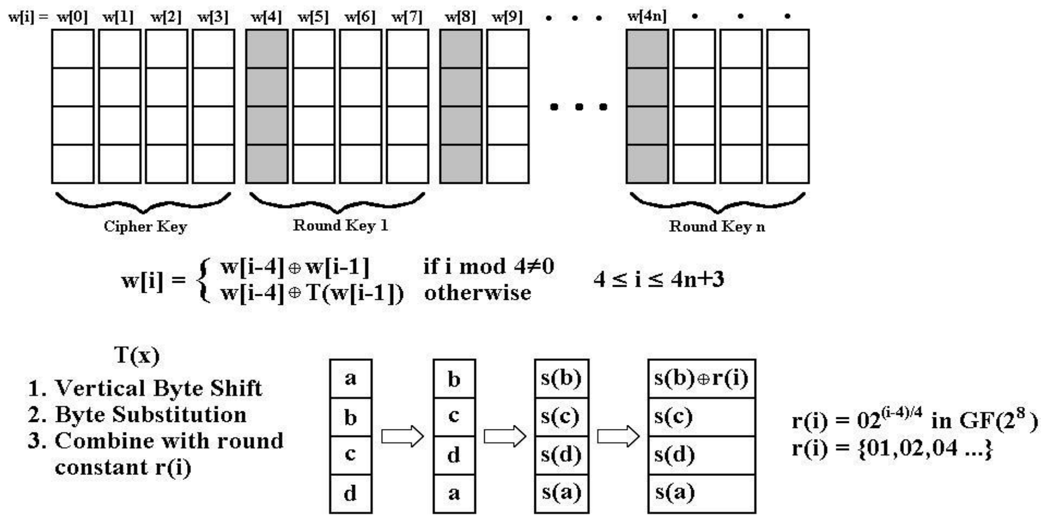

w[i] = w[0] w[1] w[2] w[3] w[4] w[5] w[6] w[7] w[8] w[9] • • • w[4n] • • •
Cipher Key
Round Key 1
Round Key n
w[i] = { w[i-4] ⊕ w[i-1] if i mod 4≠0; w[i-4] ⊕ T(w[i-1]) otherwise   4 ≤ i ≤ 4n+3
T(x)
1. Vertical Byte Shift
2. Byte Substitution
3. Combine with round constant r(i)
a b c d
b c d a
s(b) s(c) s(d) s(a)
s(b)⊕r(i) s(c) s(d) s(a)
r(i) = 02^((i-4)/4) in GF(2^8)
r(i) = {01,02,04 ...}

Figure 1.12: Key generation schedule [13]

Hence, for a 128-bit input plain text and cipher key, AES has 10 rounds and every round undergoes the steps described in this section. The repetitive operations establish a pattern in the computation which help an adversary to identify the algorithm from its power trace using SPA.

In this research, the iterative property of AES is exploited to adopt a pipelined approach to its garbled circuit implementation in hardware making it practically usable and more leakage resilient.  The detailed implementation is explained in chapter 3 and the results have been discussed in chapter 4.

# Chapter 2

# Literature Review

In Chapter 1 the preliminaries of Garbled Circuits and DPA algorithms have been discussed. This chapter focuses on peculiar aspects and limitations of the previous work on the implementation of Garbled Circuits in hardware.

## 2.1 Token implementations

Fig 2.1 illustrates a token based client-server system. The client must authenticate itself with the access token every time it needs to access the server.

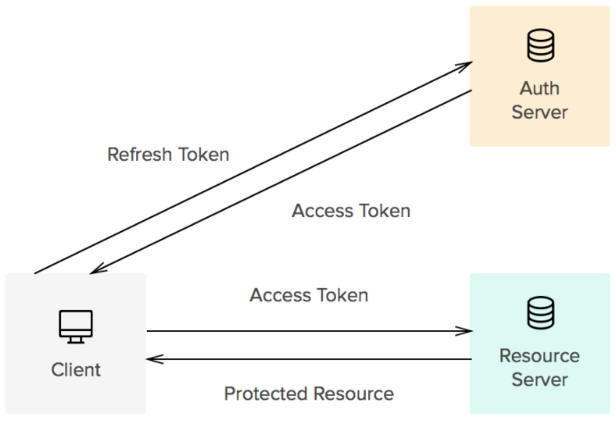


Figure 2.1: Token implementation in a typical client-server system [15]

The refresh token is a request placed by the client to the authentication server which then responds with a access token. The client uses access token to communicate with the server. Without the correct access token, the server wouldn't identify the client as a legitimate user.

Majority of the previous works on cryptographic primitives focus on use of AES-128 or DES as the function (C) to be garbled. Other circuits considered relate to higher level applications, like database search or bioinformatics, rather than tasks required of a cryptographic token. [pinkas] provide the first software based feasibility results of GC constructions. Let's delve further in to some token based implementations.

### 2.1.1 One-time Programs(OTP)

In section 1.2 (GC protocol description), it has been stated that to provide the receiver R with garbling keys for his inputs, an Oblivious Transfer(OT) is run between sender S and receiver R. The OT is hence considered to be the only interaction between S and R. In further research, it has been suggested to extend Trusted Platform Module(TPM) to implement non-interactive OT for non-interactive Circuit garbling. Hence One-Time Programs(OTP) were introduced. Instead of a semi-honest R, this approach considers malicious receivers and can be viewed simply as Yao's Garbled Circuit (GC), where the oblivious transfers (OT) are implemented with One- Time Memory (OTM) tokens. An OTM token $T_i$ is a simple tamper-proof hardware, which allows only a single query of one of the two stored garbled values. A tamper-proof one-time-settable bit bi which is set as soon as the OTM is queried can also be used. OTM-based GC execution can be non-interactive; the sender can send the GC and corresponding OTMs to the receiver, who will be able to execute one instance of SFE on any input of his choice. Finally, GC with OTP is inherently a one-time execution object (generalizable to k-time execution by repetition). [7]

The advantage of this approach is that very little protection against side-channel attacks is required in the circuit's implementation and the disadvantage is the limited number of GC executions: consider a real-fife use-case where the token is a credit card rendered unusable after k number of uses; this is not practical. [15].

### 2.1.2 Out sourcing

Consider E, a cloud computing provider; the role of sender is split between a secure hardware token GS and some other party GU (e.g., a desktop workstation). The idea is for GU to generate Bf, which is passed to GS and translated (securely) into a garbled circuit(GC). The GC can then be evaluated by E, with the overall effect of securely out-sourcing computation from GU to E. This scenario is advantageous in the sense it allows a flexible choice of f (wrt. the token) and is very speed- and memory-efficient. However, it has a relatively high hardware requirement. [15]

### 2.1.3 Observation

To the best of our knowledge almost all the previous implementations are either semi-honest models or included interaction between S and R between protocol execution [276], which precluded the receiver R from choosing his input adaptively, based on given (and even partially evaluated) garbled circuit. This possibility of adaptively chosen inputs results in possible real attacks by a malicious receiver R in the non-interactive setting. In our implementation, S and R are assumed to be non-interactive and R can be semi-honest or malicious. S is the manufacturer who makes the device and gives it to R. R can be either a naïve user who can be exploited by an adversary who is trying to guess his AES key. R does m encryption operations with his AES key and the adversary tries to apply the DPA algorithm on the power traces of the operations. In Chapter 4 this scenario is replicated and it is concluded that the current implementation though allows R to choose his inputs but is leakage resilient.

## 2.2 Memory Utilization and Design Process

One of the major limitations of previous garbled-circuit implementations is the memory required to store the entire garbled circuit in the memory, not to forget which includes the encoding logic and keys for each gate. However, there is no need, for either the circuit generator or evaluator to hold the entire circuit in memory. The garbled circuit generation and evaluation processes can be overlapped in time (pipelined), eliminating the need to store the entire GC in memory. In this research, the iterative property of AES is used to create pipelined states and reuse the existing hardware. Also, since the operations done in each round are also redundant, we can also create more pipelines along the design process to reduce the hardware density further.  However, it should be noted that pipelining and hardware reuse decreases the latency of the hardware but would deteriorate the throughput. The designer should consider this trade-off during the design process.

Majority of the previous works [16] unlike [17] are software implementations of GC but this research uses Verilog HDL and is compiled and synthesized for a cyclone IV FPGA. We have adopted an intuitive power analysis procedure that uses a combination of 3 tools in the Quartus II software to generate the power trace for analysis. The details of the implementation are discussed in chapter 3.

# Chapter 3
# Proposed Methodology

Following the deficiencies mentioned in the last chapter, this thesis focuses on reusing the garbling keys at various levels of the circuit. Also, from the iterative property of AES (discussed in chapter 1) it is intuitive that it is unnecessary to generate and store the entire circuit at once, the circuit generation is pipelined and reused which decreases the total logic elements required for implementation on an FPGA, the critical path and the latency of the execution of the algorithm and the power consumed by the FPGA; this has also been explored in [5]; however, we pipeline the individual iterations as well.

The above two approaches might seem to compromise the resilience of the implementations against DPA attacks but the concept of red and blue gates discussed in the following section intended to make it resilient to the attacks.

## 3.1 Concept of Red and Blue gates

From the discussion in section 1.2 of this thesis, it is imperative that the hardware implementation of garbled circuits requires 6 keys to implement each 2 input gate in hardware (2 for each input wire and 2 for the output wire). The bank of gates required to implement AES algorithm is divided in two sets of gates named as the red and the blue gates. Let us assume we have a 32-bit garbling key(k) for implementing a simple circuit shown below:

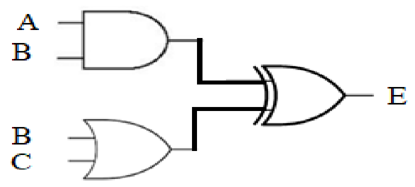


Figure 3.1: Sample circuit for implementation

key k is divided in to 4 keys k1, k2, k3, k4 of 8 bits each. Blue gates have their inputs garbled with keys k1 and k2, outputs garbled with keys k3 and k4 as shown below:

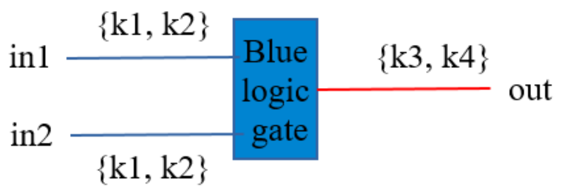


Figure 3.2: Garbled Blue gate

In the above figure, key k1 is used as the garbled representation of the bit '0' and k2 is used to represent bit '1' in both in1 and in2. Key k3 is used to represent bit '0' in the output wire and k4 is used to represent bit ‘1’. Red gates have their inputs garbled with keys k3 and k4, outputs garbled with keys k1 and k2 as shown below:

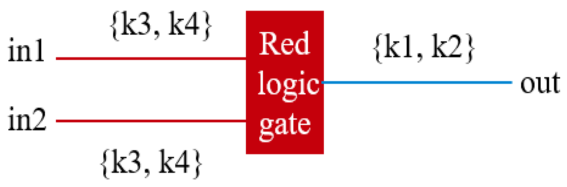


Figure 3.3: Garbled Red gate

In the above figure, key k3 is used as the garbled representation of the bit ‘0’ and k4 is used to represent bit1 in both in1 and in2. Key k1 is used to represent bit 0 in the output wire and k2 is used to represent bit ‘1’. let’s call k1 and k2 as blue keys and k3 and k4 as red keys since they represent the inputs of corresponding garbled gates.

The output of every blue gate in the circuit is fed to the input of a red gate vice-versa. Let us consider the example of garbling the sample circuit in fig 3.1.

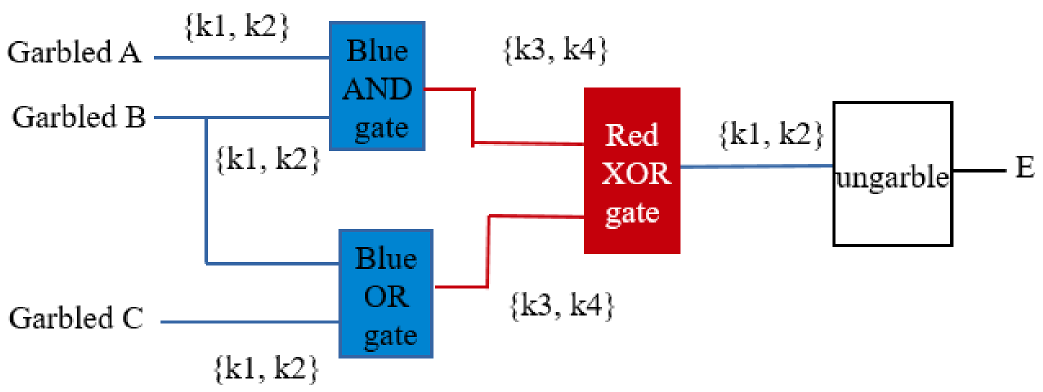


Figure 3.4: Sample circuit implemented using Garbling technique

From the above explanation of blue and red gates it should be noted that if the circuit alternates between blue and red gates, there is no conversion between red and blue input keys would be required.

## 3.2 AES implementation with blue and red gates

In Section 1.4, all the steps involved in AES algorithm implementation were discussed. To summarize; substitution, row transformation, mix-columns, add key were the steps that were being repeated in the first 9 rounds. Their garbled implementation is discussed in this section.

### 3.2.1 Process Flow

Fig 3.5 depicts the process flow of this thesis and it is followed by a brief explanation of the design procedure.

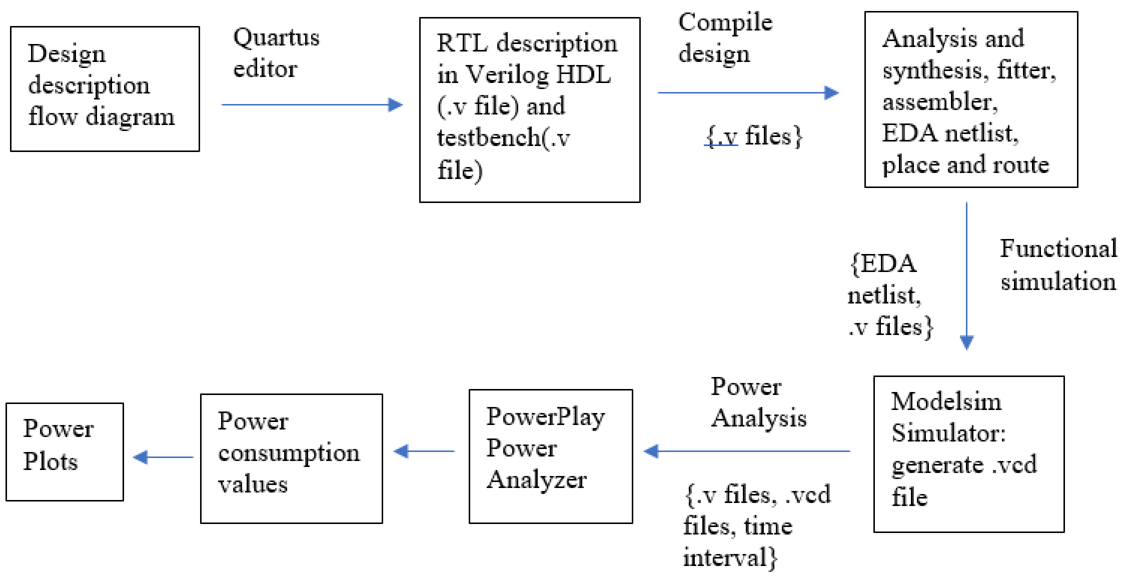

Design description flow diagram
Quartus editor
RTL description in Verilog HDL (.v file) and testbench(.v file)
Compile design
{.v files}
Analysis and synthesis, fitter, assembler, EDA netlist, place and route
{EDA netlist, .v files}
Functional simulation
Modelsim Simulator: generate .vcd file
Power Analysis
{.v files, .vcd files, time interval}
PowerPlay Power Analyzer
Power consumption values
Power Plots

Figure 3.5: Process flow followed in this research

The underlying algorithms have been discussed in detail in chapter 1. A flow diagram was first designed for control and datapaths of the design: a Finite State Machine (FSM) was created for the design with requisite number of states. The FSM design and FSM state logic was then coded using Verilog HDL in Quartus II editor. The FSM initially had 4 states; but the multiplier being a complicated operation, created a high critical path and area as compared to the other operations, this unnecessarily increased the interconnect length and the frequency of operation of the design. Hence, to reduce the critical path and number of logic elements required for the design, area intensive components like multipliers have been reused; which means that the design has been pipelined in the mixcolumns step. This has increased the number of states in the FSM to 40. Pipelining is discussed in section 3.2.2.

As discussed in chapter 2, the entire circuit need not be created when the device has been manufactured. The same hardware module can be reused at different instances(states) of the device operation with the inputs corresponding to that state. For Example: An xor module (hardware component) can be used in the mixcolumns step and then reused in the add-round-key step with different inputs. This insight reduces the hardware utilization but increases the number of FSM states and decreases the throughput the device. This should be noted by the designer and he must make sure his design meets the timing constraints placed by the consumer. Initially the xor gates and multiplier components were designed (discussed in section 3.2.3): separate .v files are created, compiled and verified. The .v files for each module should be included in the Quartus project to be able to instantiate the components whenever required. This essentially reduces the over-head of repeating the same code snippet in the main design and makes the code legible.

The main design (called the top-level module) along with all the sub modules ( .v files) is then compiled (analysis and synthesis, EDA netlist generation) for Cyclone IV GX FPGA device using quartus_map and quartus_eda. The next step is to perform a functional verification and generate a .vcd file using Modelsim simulator. A testbench file corresponding to the top-level module is written; the ‘dumpvars’ command is used in testbench file to dump all the signal activities for the current simulation in to a .vcd file which is used as an input to the Power analysis. Refer [18] for Altera-Quartus and Modelsim flow. The power analysis procedure is discussed in section 3.4 of this chapter.

### 3.2.2 Pipelining and Design reuse

Fig 3.6 and Fig 3.7 depict how the area and interconnect length of a design decrease with pipelining. Since the inputs and outputs are registered, at different FSM states we give different inputs and get different outputs, which means that though the component has been instantiated only once it has been reused decreasing the area, increasing the frequency of the design and the critical path.

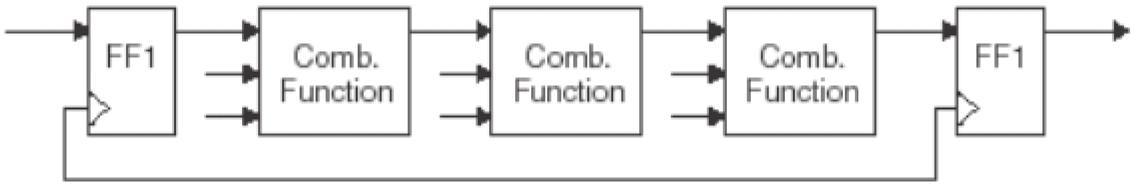


Fig 3.6: Before Pipelining [19]

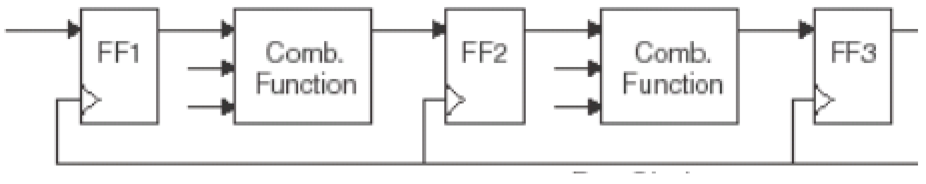


Figure 3.7: After Pipelining [19]

### 3.2.3Preliminaries

The modules for garbled blue xor gate(G(B(xor)), garbled red xor (G(R(xor)) gates are created to reuse them when ever needed. AES has GF (2) multiplication operations which can be implemented using xor gates and shift operations. The next task is to create multipliers using the above created xor gates and shifting operations. For multiply by 0x02 operation, Garbled blue(G(B(m2)) and red (G(R(m2)) components are created. For multiply by 0x03 Garbled blue (G(B(m3)) and red (G(R(m3)) components are created.

### 3.2.4AES rounds

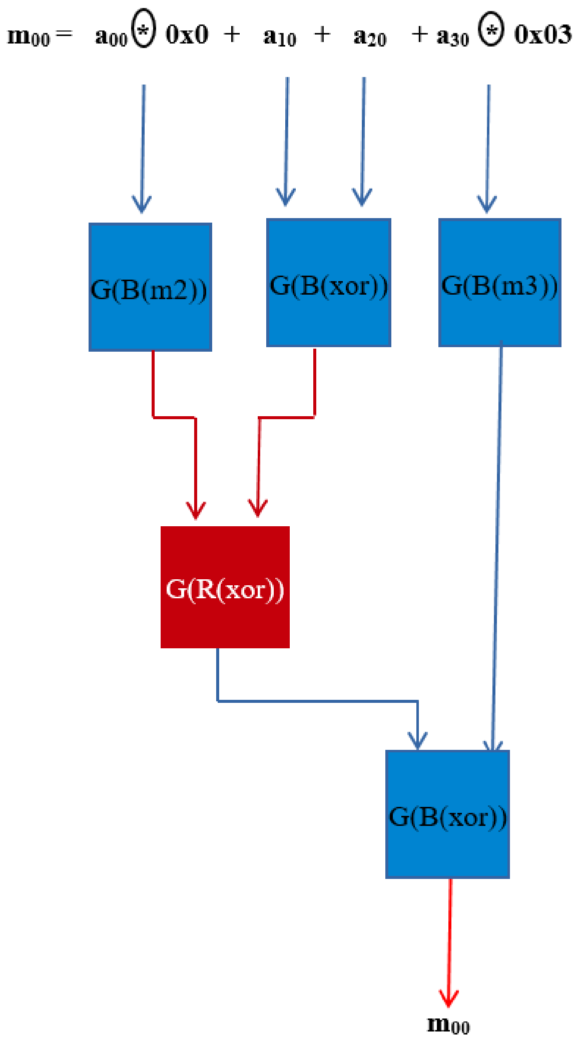
AES is a block cipher, the plain text divided in to 16 plain text is garbled using keys k1 and k2. The garbled text is then divided in to 16 blocks and stored in a 4x4 matrix like the original AES implementation. Each block is of 64-bit.

| | | | |
|---|---|---|---|
| $a_{00}$ | $a_{01}$ | $a_{02}$ | $a_{03}$ |
| $a_{10}$ | $a_{11}$ | $a_{12}$ | $a_{13}$ |
| $a_{20}$ | $a_{21}$ | $a_{22}$ | $a_{23}$ |
| $a_{30}$ | $a_{31}$ | $a_{32}$ | $a_{33}$ |

Figure 3.8: Output after row transformation

- **Round 1 to round 9**

The AES sbox is used to carry out the substitution operation followed by a row transformation. The output at this point, a 4x4 matrix garbled with keys k1 and k2 is fed as an input to the mixcolumns step. The above matrix is to be multiplied by AES multiplication matrix. In every column one block is multiplied by 0x02; (G(B(m2)) is instantiated for this operation and one block is to be multiplied by 0x03, (G(B(m3)) is instantiated for this operation. Multiplication by 0x01 returns the same block. Now, $\mathbf{a_{00}}$ is replaced by $\mathbf{m_{00}}$ and $\mathbf{a_{01}}$ by $\mathbf{m_{01}}$ and so on.



$\mathbf{m_{00}} = \mathbf{a_{00}} \circledast \mathbf{0x0} + \mathbf{a_{10}} + \mathbf{a_{20}} + \mathbf{a_{30}} \circledast \mathbf{0x03}$
G(B(m2))
G(B(xor))
G(B(m3))
G(R(xor))
G(B(xor))
$\mathbf{m_{00}}$

Figure 3.9: Summarizing the operations on column 1

Similarly, other parts of the matrix are also updated with m values which are garbled in keys k3 and k4.

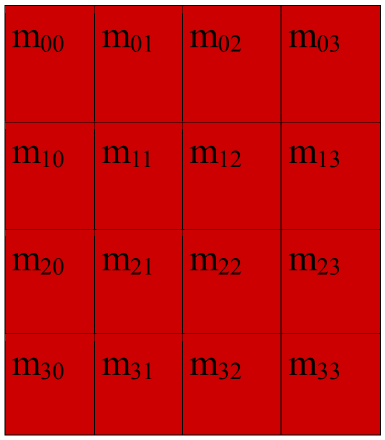

$m_{00}$ $m_{01}$ $m_{02}$ $m_{03}$
$m_{10}$ $m_{11}$ $m_{12}$ $m_{13}$
$m_{20}$ $m_{21}$ $m_{22}$ $m_{23}$
$m_{30}$ $m_{31}$ $m_{32}$ $m_{33}$

Figure 3.10: Output of mix columns step matrix m.

Key generation:

User's key is used to generate key for each round as discussed in chapter2.

The key matrix is garbled with red keys (k3 and k4) and is xored using G(R(xor)) with the above m matrix to give the sate matrix s; this matrix is the input to the next round.

- **Round 10**

In round 10, the mixcolumns step output matrix m is the desired output. So, it is ungarbled and the output is made available at the output register.

## 3.3 Power Analysis

This section focuses on various factors effecting the power consumption and then the effect of our methodology described in the previous section is discussed. The two major sources of power consumption in FPGAs are:

- Static power: It is due to leaking currents when the device is standby. It increases with a decrease in size of the transistors used to build the device. Changing the inputs wouldn't impact static power much because area of the design isn't effected.
- Dynamic Power: Dynamic power dissipation during charging and discharging of internal capacitances in the logic array and interconnect networks of an active device. Hence this depends on the design implemented o the device and the signal activity during its execution. Hence this research is uses dynamic power consumption to perform the analysis. [20]

### 3.3.1 Signal Activities

In estimating power consumption, the behavior of each signal involved the design is a primary factor. The two vital statistics used for estimation are the toggle rate and the static probability.

The toggle rate of a signal can be defined as the average number of times that signal changes its value per unit of time. The units for toggle rate being transitions per second, where a transition is a change from 1 to 0 or 0 to 1. The static probability is the fraction of time the signal hold logic 1 during the period of device is being analyzed.

### 3.3.2 PowerPlay Power Analyzer: Power Analysis

When the FPGA design is complete, accurate device power consumption can be estimated with the PowerPlay Power Analyzer tool in Quartus II software. The PowerPlay Power Analyzer tool provides a flexible interface for specifying signal activities, which reflects the critical importance of using representative signal activity data from the .vcd files during power analysis. Therefore, data sources used are simulation results (.vcd files); user-entered node, entity and clock assignments, user-entered default toggle rate assignment.

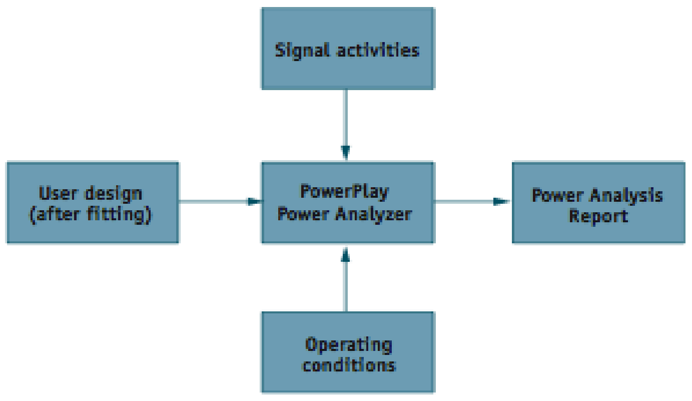
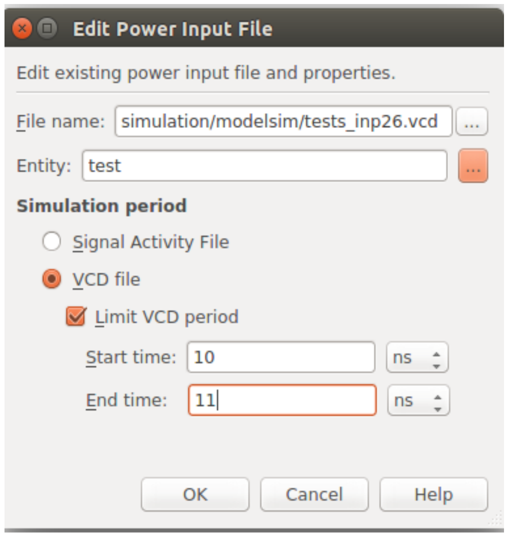


Figure 3.11: High-level Power Analyzer Flow [20]

The Power Analyzer lets designers mix and match signal activity data sources on a signal-by-signal basis. Intel-Altera claims that using simulation results is the most accurate way to generate signal activities [20]. This, in turn, is used to accurately estimate the power consumed by the device during its operation. This flow provides the highest accuracy, as all node activities reflect actual design behavior, provided that supplied-input vectors are representative of typical design operation.

The .vcd file is added as an input file for the Power analyzer. However, analyzer at this point will only report the average power consumption for the entire simulation time; to perform a power analysis a data set with high sampling rate is required. Quartus II provides an option to limit the vcd period in the power play settings dialogue box. A screen shot is shown below:



Edit existing power input file and properties.

File name: simulation/modelsim/tests_inp26.vcd

Entity: test

**Simulation period**

( ) Signal Activity File

(●) VCD file

[✓] Limit VCD period

Start time: 10 ns

End time: 11 ns

Figure 3.12: Powerplay .vcd period limit screen shot

In this research, the .vcd period is limited to 1 ns interval, the power analyzer report is generated and the dynamic power consumption values are noted.

### 3.3.3 Trace generation

It has been discussed in chapter 1 that for DPA, power traces T for m input plain text samples and the process is repeated with different AES keys and garbling keys for the analysis. Generating power reports for every ns interval is indeed a herculean task and hence in this research the process was automated and this section provides an insight on the process automation.

An Ubuntu 16.04 system has been used with Quartus and Modelsim installed in it. Quartus also has a command line interface and tcl scripting option. However, the power analyzer command line doesn't have command line option for limiting the .vcd period. Therefore, keyboard and mouse automation has been done to using an in-built tool called the 'xdotool'; a bash script is written to automate the .vcd file selection, .vcd time period, starting the power analysis process, entering the power consumption values in to the spreadsheet, incrementing the vcd period in the dialogue box shown in figure and starting the analysis again. The start time and end time of the vcd period are incremented by 1ns in every repetition till the end time is equal to the end of vcd period. The values stored in the spreadsheet are then plotted to generate the power traces for that simulation. If a windows system is to be used, the automation can be done in python using the pywinauto package.

The above procedure has been used to generate power traces T for m input plain text samples with different AES input keys and Garbling keys as required. The traces are then used for analysis and the analysis and results are described in section 5.

# Chapter 4
# Results and Discussion

This chapter focuses on the Differential Power Analysis (DPA) results generated using the methodology described in chapter 3, the basic idea of DPA has been discussed in chapter 1 The following section discusses the implementation.

## 4.1 Differential Power Analysis on AES key

DPA analysis has the following steps: Simple Power Analysis, Record encryption m Power traces, align traces based on Selection bit, Calculate and store the differential trace, use an AES key guess and record m Power traces again with this key, sort the traces in to their respective sets marked in the previous analysis. If the key guess was correct, the differential of the differential traces would look like the actual trace else there would be unexpected spikes in the aforementioned trace.

**Simple Power Analysis(SPA):** Most basic form of Power Analysis. The attacker would be able to identify the algorithm being used in the encryption based on the SPA trace. The presence of garbling makes SPA almost impossible, but in this research, it is assumed that the attacker has a knowledge of the algorithm being used(malicious Receiver R).

**Differential Power Analysis(DPA):** This is a more powerful and detailed attack and uses a statistical approach as discussed in chapter1. m samples of random input plain texts have been used for analysis (m=10 in this research). In each analysis, the output matrix of the 1st round of AES is monitored. As mentioned in chapter 1, a bit b is monitored form the output state matrix to sort the traces in to set 0 and set 1, the DPA selection function D(C, b, Ks) is used to calculate the value of a bit b for a given circuit C and AES key Ks; the 96th bit of the output state matrix (a random assumption) has been used as bit b in this research. However, the simulation software Modelsim has an option called the wave window where the values stored in a register/wire during any point of the simulation process can be viewed. Hence instead of calculating a function D(C, b, Ks), the values of bit b have been noted at the end of each simulation.

### 4.1.1 Power trace with Original keys

AES key K1(original AES key) and garbling key gk1 are used in this step. For each of the ten inputs keys K1 and gk1 and generate power traces.(T1...T10). Based on the bit b the traces are sorted in to two sets: set 1 (T1, T4, T6, T8) where the bit was '1' during the computation and set 0 (T2, T3, T5, T7, T9, T10) where the bit was '0' during the computation. Then average the traces in each set and generate average trace for set 0 and average trace for set 1.

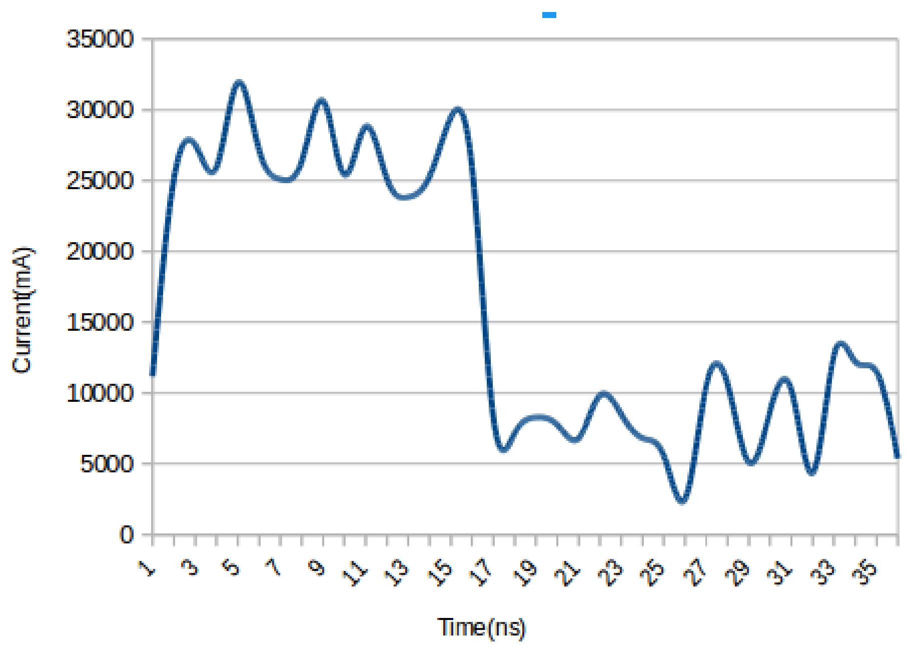


Figure 4.1: Set 0 average Power trace (gk1 and K1) keys

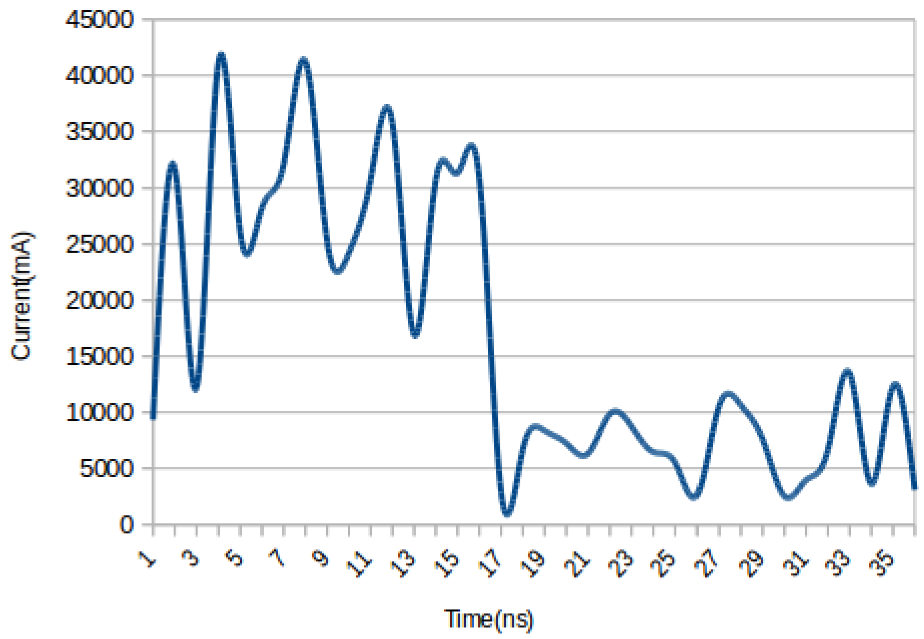


Figure 4.2: Set 1 average Power trace (gk1 and K1) keys

Differential trace for set 0 (Fig: 4.1) and set 1 (Fig: 4.2) traces is plotted

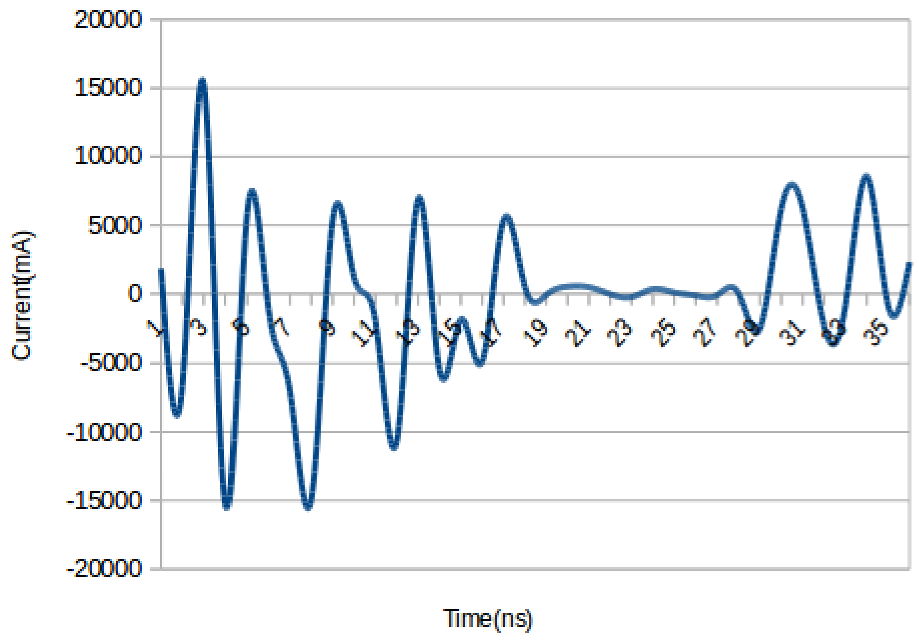


Figure 4.3: Differential Trace of Fig: 4.1 and Fig: 4.2 traces for (gk1 and K1) keys

### 4.1.2 Power trace for AES key guess

The above analysis is repeated with AES key K2 and garbling key gk1 to generate power traces, T`1, T`2 …. T`10. Note that K2 and K1 differ in just one bit (108th bit) which is a random choice. This time the bit is not used to sort them in to sets: they are placed in to the same sets as in step 1. The average power traces for set 0 and set 1 are calculated.

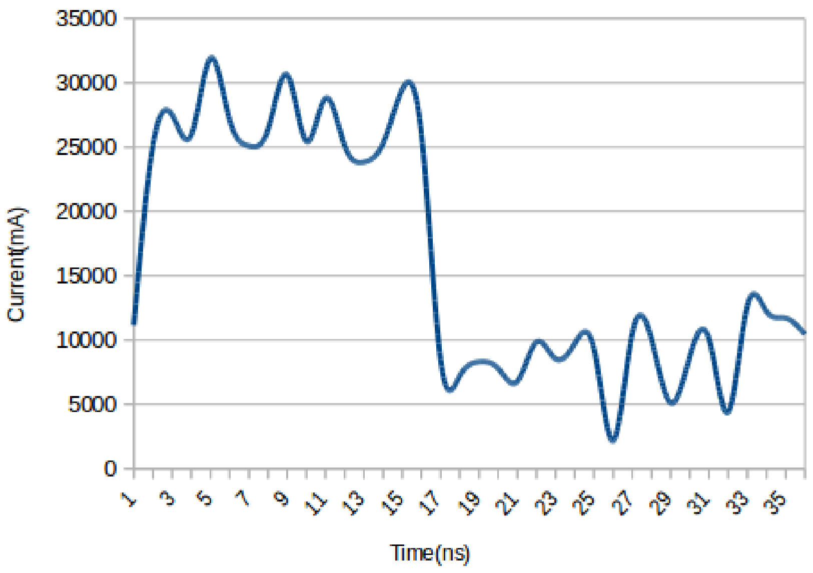


Figure 4.4: Set 0 average Power trace (gk1 and K2) keys

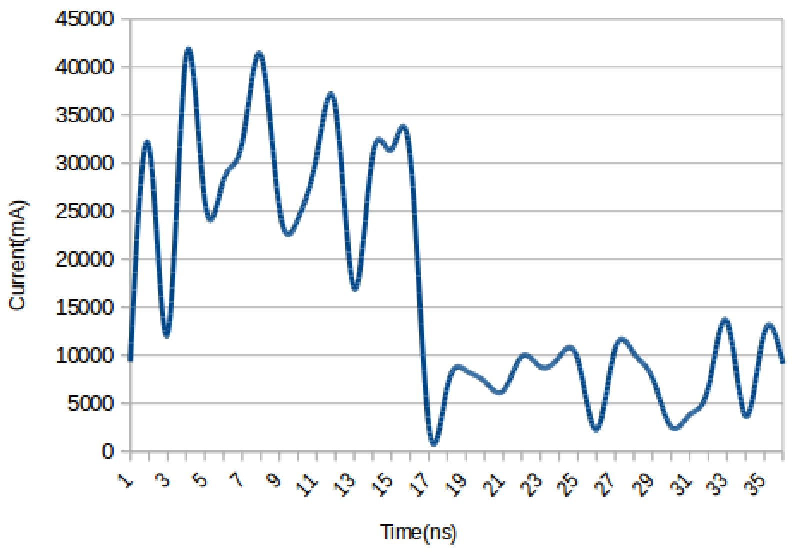


Figure 4.5: Set 1 average Power trace (gk1 and K2) keys

Differential trace for set 0 (Fig: 4.4) and set 1 (Fig: 4.5) traces is plotted.

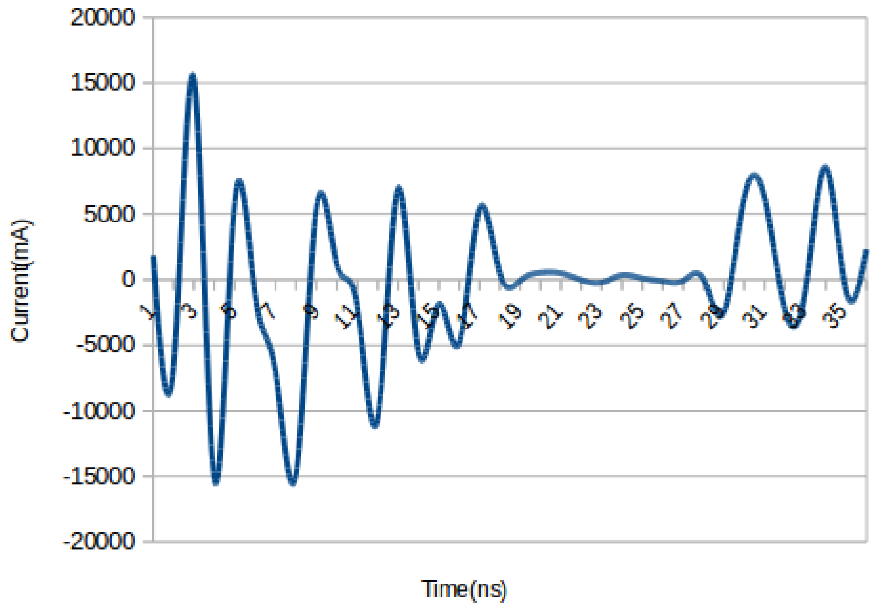


Figure 4.6: Differential Trace of Fig: 4.4 and Fig: 4.5 traces with (gk1 and K2) keys

It would be difficult to observe and analyze the difference in Fig 4.3 and Fig 4.6 at a glance; therefore, a differential trace of the above two differential traces is plotted in Fig: 4.7 to observe the difference in the instantaneous power values. Ideally if K2 and K1 were identical, this differential (Fig: 4.7) should have no peaks since the trace then corresponds to the difference of identical values.

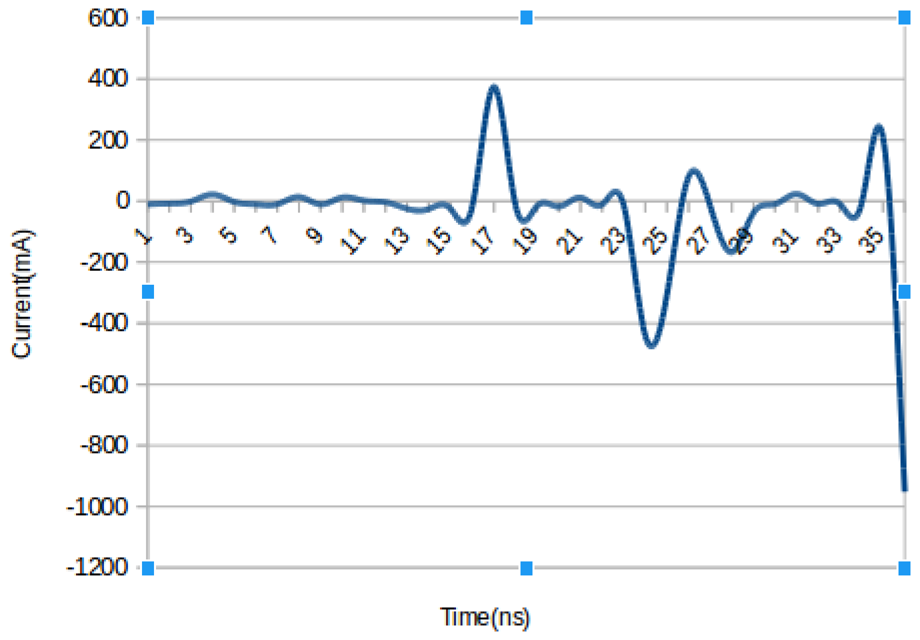


Figure 4.7: Differential of differential traces K1, K2 AES keys and gk1 garbling key

The peaks indicate that the keys K1 and K2 were different it would be an easy guess for an attacker to know that his key guess was wrong and when he toggles the 108th bit which he was trying to guess, the differential trace wouldn't have those peaks and hence he gets to guess the correct key value. For a comparison, the above analysis has been repeated with another key K3 where K3 and K1 differ in two bits (108th bit and 126th bit again a random choice). The differential of the differential traces is plotted in Fig:4.8 and these peaks are evidently higher than the peaks in Fig: 4.7. Hence, an intelligent attacker would toggle his 126th bit, re-do the analysis and then he would have a differential similar to Fig: 4.7 and toggles the 108th bit to get the AES key even without having access to the device.

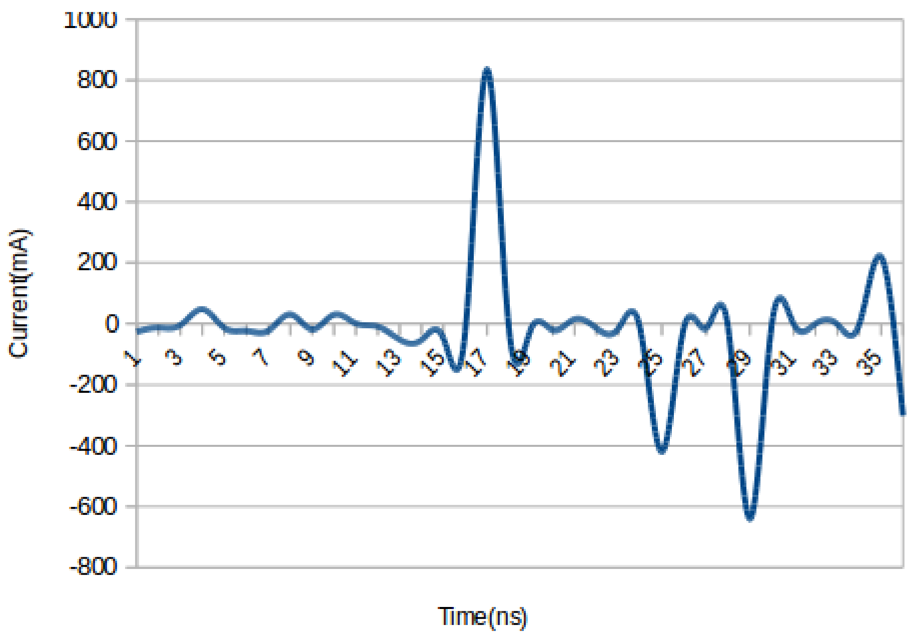


Figure 4.8: differential for differential traces K1, K3 AES keys and gk1 garbling key

## 4.2 Differential Power Analysis on Garbling key

This section explores a possibility of the attacker trying to guess the garbling key.

### 4.2.1 DPA on garbling key without reusability

In 4.2.1 and 4.2.2, we assume the scenario of a malicious user, he knows his AES key (worst case scenario) and is trying to guess garbling key to get a fair idea of the system implementation and may be reverse engineer! Also, it should be noted that garbling has been implemented without OTPs.

In 4.2.1, we consider the scenario where the red and blue gate concept is not used, i.e., no reusability and the circuit construction is similar to the concept in Fig 11. In this case a long garbling key is used and each wire in the circuit has unique garbling keys. Changing the design to remove reusability might create arbitrary results in a comparative analysis with our original DPA trace (fig 20). In our approach to key reusability, we have a garbling key gk1 that repeats itself to garble every gate in the circuit hence the garbling key for the circuit we used in section A is actually a large key which has 32-bit gk1 being repeated periodically ({gk1, gk1, gk1...gk1}). In thisanalysis, we need a garbling key whose hamming distance is 1 from the original long key. We hence replace gk1 with gk2 in the last operation (add-round key), the first 3 operations have gk1. Essentially the analysis is now between the traces generated with **key1= {gk1, gk1,…gk1}** and **key2 ={gk1, gk1,…gk2}** where key2 and key1 differ only by 1 bit. Fig 22 is the differential trace for the current scenario and Fig 20. It is evident that the spikes are only in the last part of the trace where gk2 is used for garbling. IT would be an easy guess to attacker to know the ambiguity and replace the corresponding bit in key2 to make it similar to key 1.

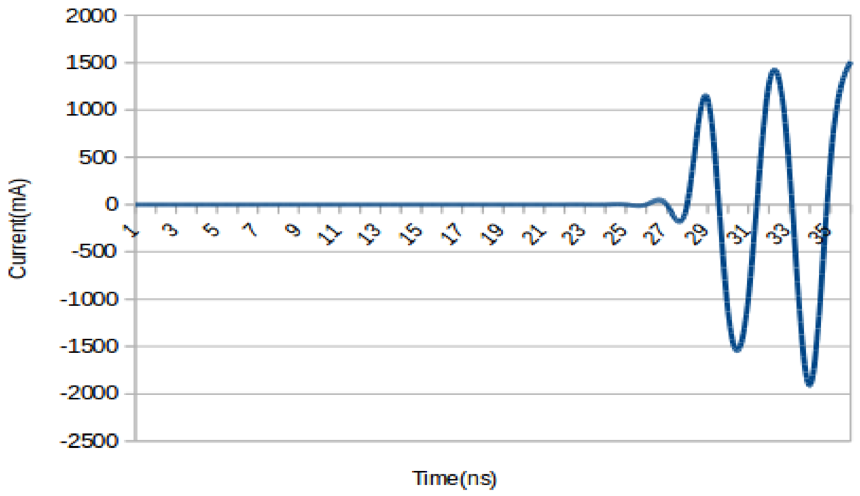


Figure 4.9: Differential trace for key 2 and key 1

A spike would be seen where ever the key is used and the attacker would know that the hamming distance between GK1 and GK2 is > 0 and he toggles the corresponding bit. Hence in this case, attack has been successful.

## 4.2.2 DPA on garbling key with reusability: using red-blue concept

In 4.2.2, we use our red blue gates for reusability and change garbling key by 1-bit: we use gk2 to garble the circuit. Hence, AES key K1 (original AES key) and garbling key guess gk2 are used in this step (reusability requires us to use gk2 for all the operations rather than using it in 1 operation). For each of the ten inputs, keys K1 and gk2 and generate power traces. (T``1, T``2…T``10). The traces are sorted in to set 1 and set 0 based on section 4.1.1; 1.e., set 1 (T``1, T``4, T``6, T``8) and set 0 (T``2, T``3, T``5, T``7, T``9, T``10). Then average the traces in each set and generate average trace for set 0 and average trace for set 1.

AES key K1(original AES key) and garbling key guess gk2 are used in this step. For each of the ten inputs, keys K1 and gk2 and generate power traces. (T``1, T``2...T``10). The traces are sorted in to set 1 and set 0 based on section 4.1.1; 1.e., set 1 (T``1, T``4, T``6, T``8) and set 0 (T``2, T``3, T``5, T``7, T``9, T``10). Then average the traces in each set and generate average trace for set 0 and average trace for set 1. It should be noted that gk2 differs from gk1 by just 1-bit (LSB).

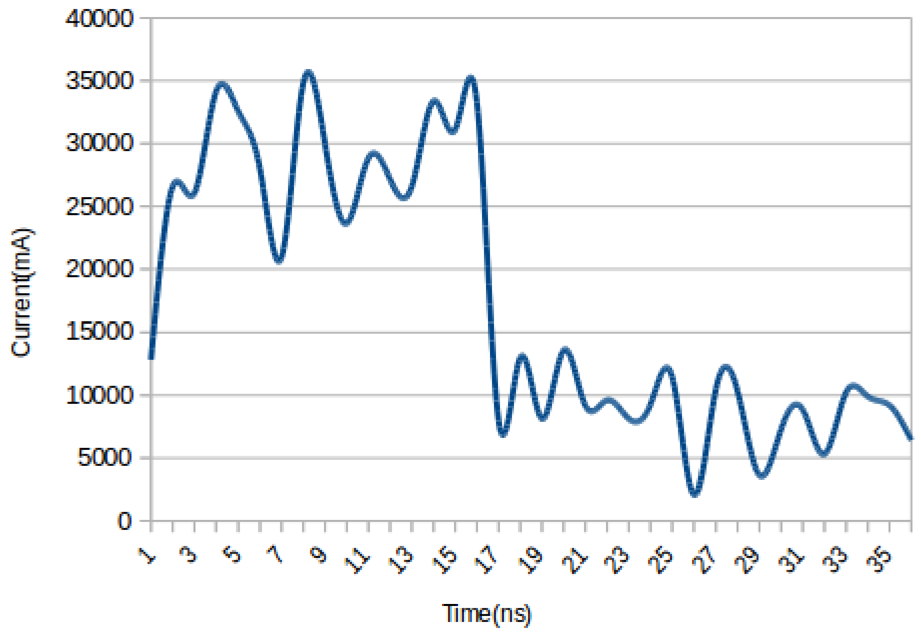


Figure 4.10: Set 0 average Power trace (gk2 and K1) keys

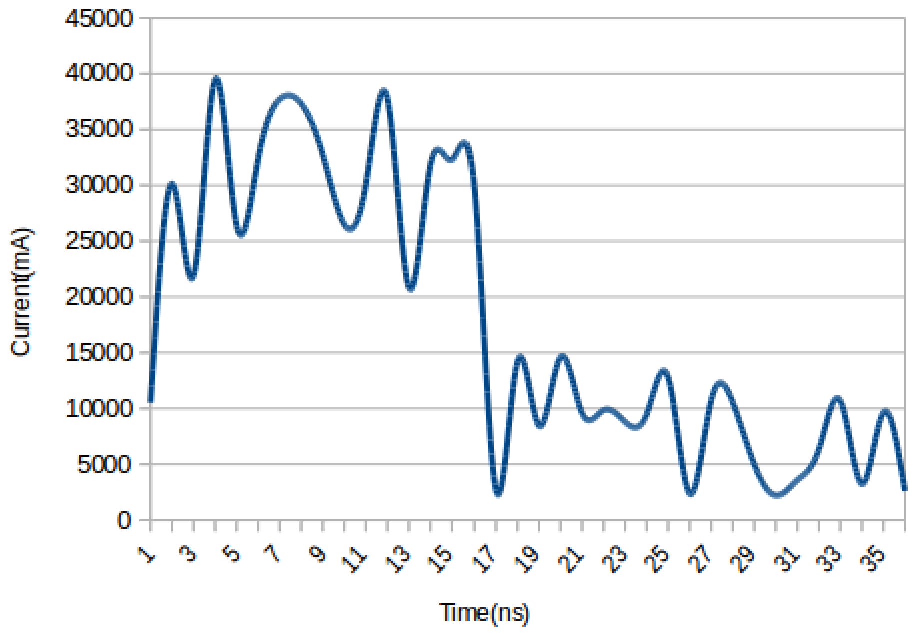


Figure 4.11: Set 1 average Power trace (gk2 and K1 ) keys

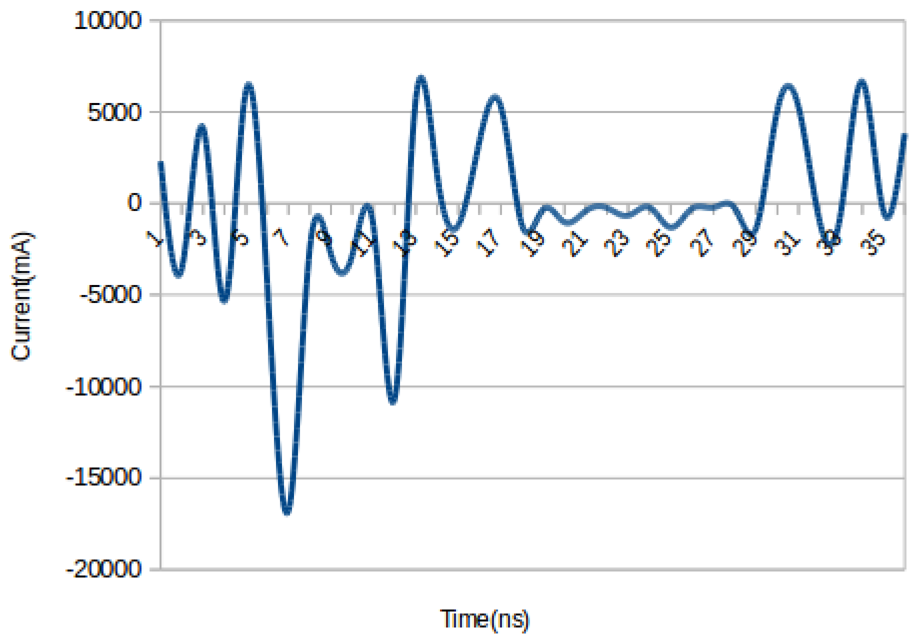


Figure 4.12: Differential Trace of the set 0 and set 1 traces (gk2 and K1) keys

A differential trace of the above two differential traces 4.11 and original differential 4.3 is plotted in Fig: 4.12 to observe the difference in the instantaneous power values.

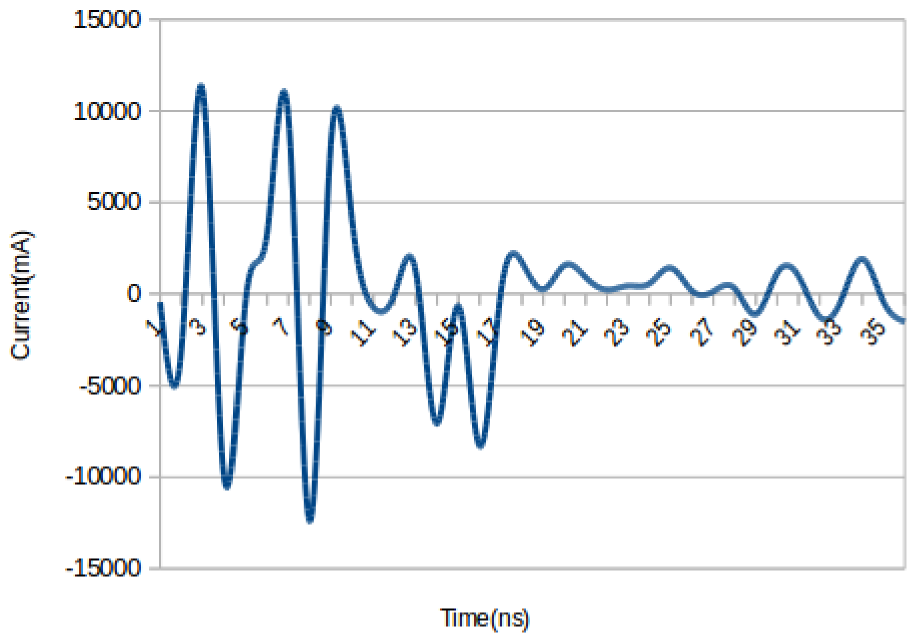


Figure 4.13: differential for differential traces K1 AES key and gk1 and gk2 garbling keys

It is evident that for a 1-bit change the peaks are all scattered owing to the reusability of the garbling key amongst the red and blue gates. A similar analysis is repeated with gk3 which differs from gk1 by 3 bits (random choice). It can be observed that the plot is again scattered and there is no correlation with the number of bits changed. Hence, even a statistical analysis like DPA fails to identify which key is closer to the actual key.

In the current analysis, it has been assumed that the attacker has a knowledge of the AES key and must guess the garbling key. However, in real world the problem would be to guess the AES key as well as the garbling key; from the above analysis, it is imperative that the presence of a garbling key that is varied periodically would result in an arbitrary differential trace. This adds up a new level of complexity to DPA: to guess the garbling key and then guess the AES key.

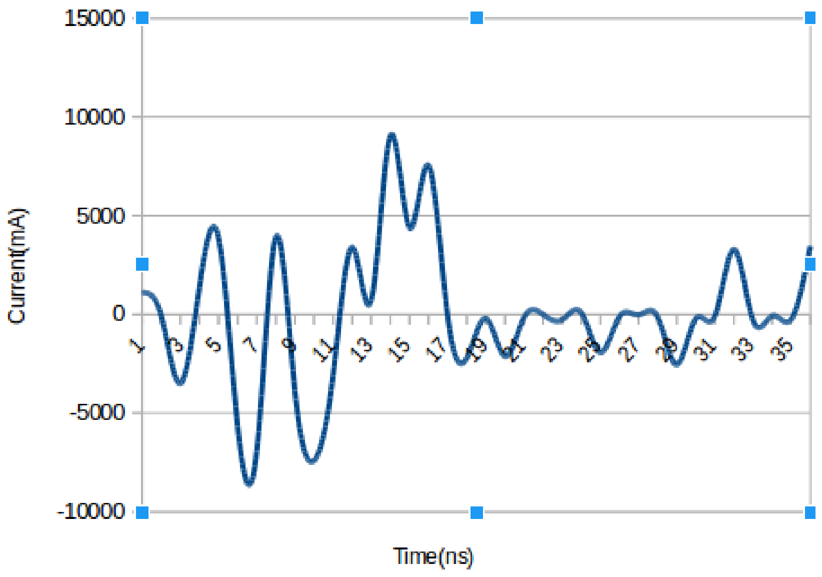


Figure 4.14: differential for differential traces K1 AES key and gk1 and gk3 garbling keys

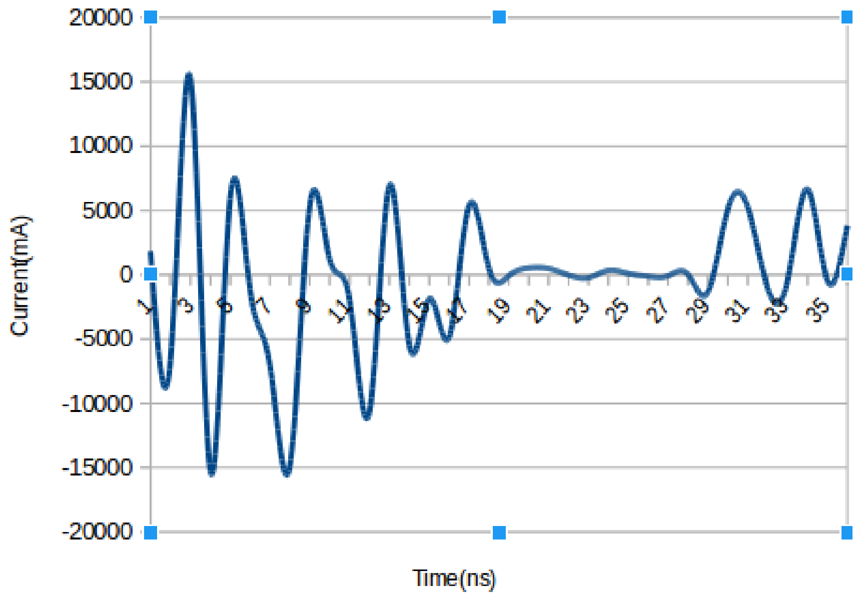


Figure 4.15: Differential trace set 0 and set 1 for gk2 and gk1 garbling keys AES key K1

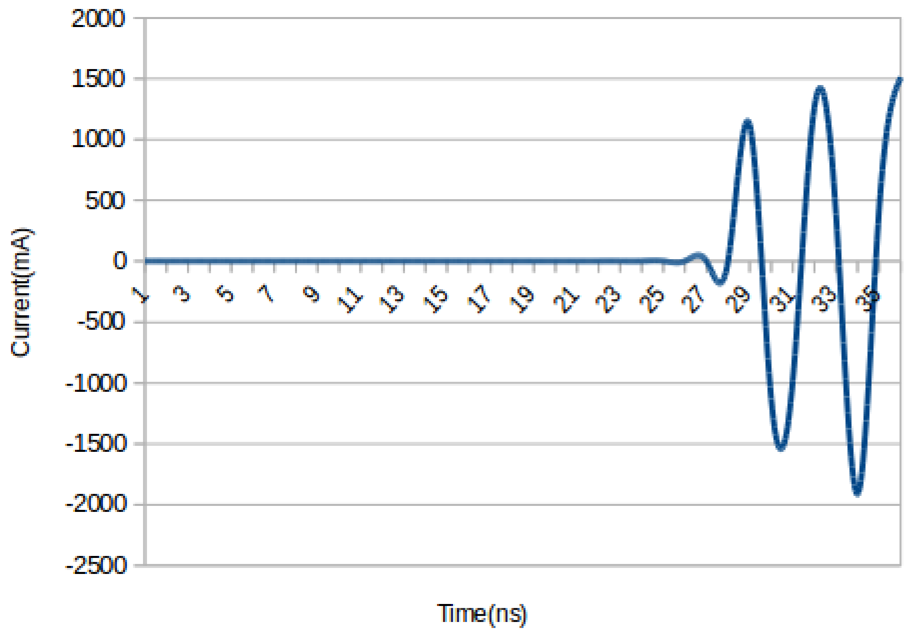


Figure 4.16: Differential of the differential traces in figure 4.14 and figure 4.3

## 4.3 Conclusion and Future Work

This research hence establishes a proof of concept that the garbling key when changed at run-time, randomizes the power trace, owing to its reusability in accordance with the concept of red and blue gates.

Parameterized Verilog modules have been used and the garbling keys were manually changed in the testbench file before every simulation to generate the value change dump ( .vcd) files. But a key generation circuit needs to be added to the system that would automatically refresh the garbling key periodically. There are several ways to implement this in a simple hardware which can be inferred from [21].

This research was focused on the implementing the idea and generating the functional simulation results. Its performance in hardware over a period of time is yet to be validated. The process of generating power traces was software-based. Though Altera-Quartus claims the tool to be accurate, it is recommended to perform the analysis using actual devices (FPGA)s and Oscillospes to observe and capture the power trace as explained in [22].

# Bibliography

[1] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis. Proceedings of CRYPTO ’99, *Lecture Notes in Computer Science*, vol. 1666, Springer, pp. 388–397, 1999.’

[2] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *Cryptographic Hardware and Embedded Systems* (CHES'01), volume 2162 of LNCS, pages 251{261. Springer, 2001.

[3] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side channel cryptanalysis of product ciphers. In *European Sumposium on Research in Computer Security* (ESORICS '98), volume 1485 of LNCS, pages 97{110. Springer, 1998.

[4] C. Cachin, J. Camenisch, J. Kilian, and J. Muller. One-round secure computation and secure autonomous mobile agents. In *International Colloquium on Automata, Languages and Programming* (ICALP'00), volume 1853 of LNCS, pages 512{523. Springer, 2000.

[5] Yan Huang. Faster Secure Two-Party Computation Using Garbled Circuits.

[6] Elisabeth Oswald. Power analysis attacks. *Crypto Group*, University of Bristol.

[7] Kimmo Jarvinen and Vladimir Kolesnikov. Garbled Circuits for Leakage-Resilience: Hardware Implementation and Evaluation of One-Time Programs. In *Cryptographic Hardware and Embedded Systems.* CHES, 2010.

[8] Vladimir Kolesnikov and Thomas Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. *Bell Laboratories*.

[9] Kevin Meritt. Differential Power Analysis Attacks on AES. *Cryptography II*. VCSG-706, May, 2012.

[10] F.-X. Standaert, Introduction to Side-Channel Attacks. In *Secure Integrated Circuits and Systems*, pp. 27–44, Springer, 2009.

[11] K. Tiri, M. Akmal, I. Verbauwhede. A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards. In *Proc. Eur. Solid-State Circuits Conf.* (ESSCIRC), Florence, Italy, 2002, pp. 403–406.

[12] K. Smith Jr., Methodologies for power analysis attacks on hardware implementations of AES. *Master's thesis, Rochester Institute of Technology*, 2009.

[13] National Institute of Standards and Technology (NIST) of U.S. Department of Commerce. *FIPS 197: Advanced Encryption Standard*, Nov. 2001.

[14] [online] https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/

[15] S. Hoerder, K. Jarvinen and D. Page. On secure embedded token design. In *Information Security Theory and Practice* (WISTP), 2013.

[16] Dahlia Malkhi, Nosam Nisan, Benny Pinkas and Yaron Sella. Fairplay- Secure Two-Party Computation system. In *USENIX Security,* pages 287-302. USENIX, 2004.

[17] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, Thomas Schneider and Farinaz Koushanfar. Tinygarble: Highly compressible and scalable sequential Circuits.

[18] Altera. Getting Started with Quartus II Simulation Using the ModelSim-Altera Software. *User-guide*. June,2011.

[19] [online] http://web.mit.edu/6.111/www/f2016/handouts/L09.pdf

[20] [online] https://www.altera.com.cn/zh_CN/pdfs/literature/hb/qts/qts_qii53013.pdf

[21] William H; Teukolsky, Saul A; Flannery, Brian P; Vettering, William T. Numerical recipes in C++: the art of scientific computing. *Cambridge University Press*, 2002.

[22] Michal Varchola and Milos Drutarovsky. The Differential Power Analysis Laboratory Setup. *IEEE Brno Czech Republic*. June, 2012.