B. TECH. PROJECT REPORT

On

Timing-based side-channel attack on Hummingbird block cipher

BY

Khushboo Sharma

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

December 2017

Timing-based side-channel attack on Hummingbird block cipher

A PROJECT REPORT

Submitted in partial fulfillment of the
requirements for the award of the degrees
of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

Submitted by:
Khushboo Sharma

Guided by:
Dr. Bodhisatwa Mazumdar
Assistant Professor

INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2017

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **"Timing-based side-channel attack on Hummingbird block cipher"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in 'Computer Science and Engineering' completed under the supervision of **Dr. Bodhisatwa Mazumdar, Assistant Professor, Discipline of Computer Science and Engineering,** IIT Indore is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Khushboo Sharma

(140001015)

CERTIFICATE by BTP Guide

It is certified that the above statement made by the student is correct to the best of my knowledge.

Dr. Bodhisatwa Mazumdar

Assistant Professor

Discipline of Computer Science and Engineering

Preface

This report on ”Timing-based side-channel attack on Hummingbird block cipher” is prepared under the guidance of Dr. Bodhisatwa Mazumdar.

Through this report I have tried to give a detailed description of timing analysis performed by me on the Hummmingbird cryptographic algorithm along with proposed methods of launching a timing-based side-channel attack.

I have tried to the best of my ability and knowledge to explain the content in a lucid manner. I have also added tables and figures to make it more illustrative.

Khushboo Sharma
B. Tech, IV Year
Discipline of Computer Science and Engineering
IIT Indore

Acknowledgements

I wish to thank Dr. Bodhisatwa Mazumdar for his kind support and valuable guidance. He was always there for us. It was his dedication and hardwork that paid off and the project was completed. It is his help and support, due to which I became able to complete the analysis and technical report. Without his advice this report would not have been possible. I cannot thank him enough.
However, this acknowledgement would be incomplete without the mention of my very able and talented partner Varsha Dhakad. Her fervent zeal and enthusiasm towards the work never failed to hearten me in all my blue moments. I could never have dreamed of completing all this work without her.

Khushboo Sharma
B. Tech, IV Year
Discipline of Computer Science and Engineering
IIT Indore

Abstract

The Hummingbird algorithm is applied in secure systems like RFID tags, wireless sensor nodes and smart integrated circuits. Hummingbird is a cryptographic algorithm which consists of a block cipher of 256-bit key and encrypts 16-bit data in one operation. It is lightweight, fast and is resistant to the most common cryptanalysis attacks like linear and differential cryptanalysis. In this report, we attempt to launch a timing-based side-channel attack on the Hummingbird block cipher. A timing-based attack takes into account the relationship between input to a cryptosystem and the time required for operations to be performed. We focus on the first subround of the first of the four block ciphers in the algorithm.

Software implementation of the subround is targeted first. Analysis is done on the dependence of execution time on plaintext, key and output of the first subround. On finding that execution time is not a good measure for distinguishing between keys due to its highly inconsistent value, a switch is made towards weighted execution time, measured by recording the execution time 20000 times for the same plaintext and key. Key search space is characterized on the basis of the number of plaintexts satisfying the relation, plaintext = output. Profiling and attack algorithms are described, analysis being done first on a smaller set of the entire key search space and then gradually increasing the size of the set. The efficiency of the attack algorithms is measured by recording the size of the residual key space and the number of times expected key is an element of the residual key space out of 100 times. A couple of miscellaneous experiments relating to sensitivity of execution time to each bit of the plaintext

and the measurement of probability of certain linear combinations of plaintexts and output being 0 are also mentioned, which posed interesting observations but could not be extended due to lack of resources and time.

Next, a hardware implementation of the subround coded in VHDL is studied. Timing simulation is performed on this design to record the time at which output is obtained. The dependency of this time on the basis of Hamming Weight of plaintext, Hamming Weight of key and Hamming Distance between initial and final values of output is investigated. This is also done wrt. the Hamming Distance between plaintext and output. Another attack algorithm is proposed based on the results and its efficiency is measured by measuring the size of the residual key space.

Finally, results of the timing analyses and the efficiency of the corresponding attacks are discussed. Conclusions drawn from the project and the scope of future work is emphasized.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Resource Constrained Devices and Lightweight Cryptography

With the advent of pervasive computing, various smart devices such as RFID tags, smart cards, and wireless sensor nodes are penetrating into and impacting people's life at a staggering rate and in significant ways. Their applications range from access control and supply-chain management to home automation and healthcare. Since a multitude of applications involve processing of sensitive personal information like health or biomedical data, the increasing demand for integrating cryptographic functions into embedded applications has risen. However, these pervasive smart devices usually have extremely constrained resources in terms of computational capabilities, memory, and power supply. For constrained devices such as these with their harsh limitations with respect to gate count and power consumption, standard cryptographic algorithms can be too big, too slow or too energy-consuming. Hence, it is desirable to employ lightweight and specialized cryptographic algorithms for many security applications. The field of lightweight cryptography focuses exactly on this.

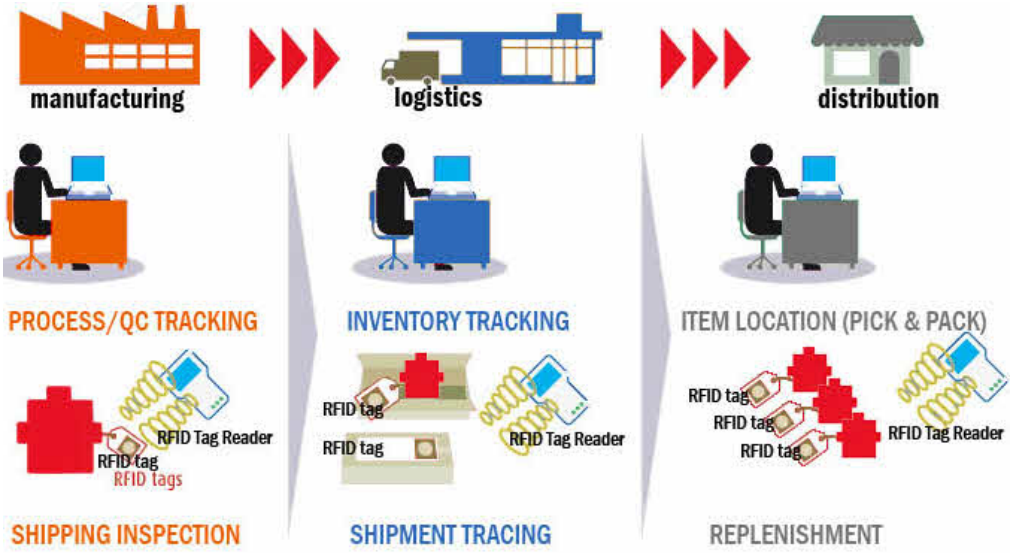

Figure 1.1: RFID tags: applications

Lightweight cryptography (LWC) is a research field that has developed in recent years and focuses in designing schemes for devices with constrained capabilities in power supply, connectivity, hardware and software. Schemes proposed include hardware designs, which are typically considered more suitable for ultra-constrained devices, as well as software and hybrid implementations for lightweight devices. Algorithms are designed such that they are capable to run on devices with very low computing power. One such algorithm is Hummingbird.

1.2 Hummingbird cipher

The overall structure of the Hummingbird encryption algorithm consists of four 16-bit block ciphers E_{k_1}, E_{k_2}, E_{k_3} and E_{k_4}, four 16-bit internal state registers RS1, RS2, RS3 and RS4, and a 16-stage LFSR. The 256-bit secret key K is divided into four 64-bit subkeys k_1, k_2, k_3 and k_4 which are used in the four block ciphers respectively. A 16-bit plaintext block PT_i is encrypted by first executing a modulo 2^{16} addition of PT_i and the content of the first internal state register RS1. The result of the addition is then encrypted by the first block

cipher E_{k_1}. This procedure is repeated in a similar manner for another three times and the output of E_{k_4} is the corresponding ciphertext PT_i.

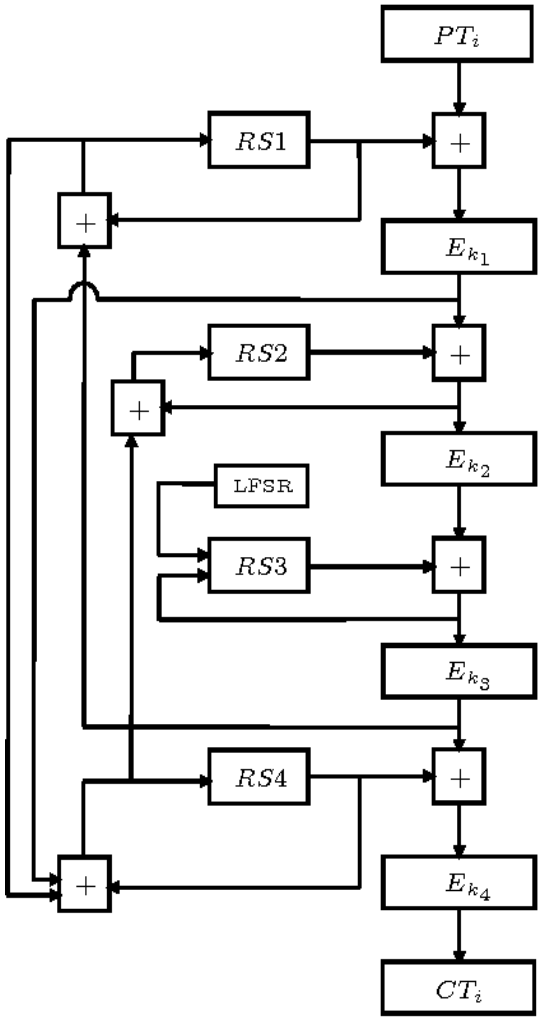

Figure 1.2: Encryption process: Hummingbird

Four identical 16-bit block ciphers are employed in a consecutive manner in the Hummingbird encryption scheme. The 16-bit block cipher is a typical substitution-permutation(SP) network with 16-bit block size and 64-bit key as shown in Figure 1.3. It consists of four regular rounds and a final round that only includes the key mixing and the S-box substitution steps. The 64-bit sub-key k_i is split into four 16-bit round keys $K_1^{(i)}$,$K_2^{(i)}$,$K_3^{(i)}$ and $K_4^{(i)}$ which are used in the four regular rounds respectively. Moreover, the final round utilizes two keys $K_5^{(i)}$ and $K_6^{(i)}$ directly derived from the four round keys. Like any

other SP network, one regular round comprises of three stages: a key mixing step, a substitution layer, and a permutation layer. For the key mixing, a simple exclusive-OR operation is used in this 16-bit block cipher for efficient implementation in both software and hardware. The substitution layer is composed of 4 Serpent-type S-boxes with 4-bit inputs and 4-bit outputs, the action of which in hexadecimal notation is described in Figure 1.3. The permutation layer in this 16-bit block cipher is given by the linear transform $L : \{0,1\}^{16} \rightarrow \{0,1\}^{16}$ defined as follows:

$$L(m) = m \oplus (m \lll 6) \oplus (m \lll 10)$$

where $m = (m_0, m_1, ..., m_{15})$ is a 16-bit data block.

Four S-Boxes Given in Hexadecimal Notation

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S_1(x)$	8	6	5	F	1	C	A	9	E	B	2	4	7	0	D	3
$S_2(x)$	0	7	E	1	5	B	8	2	3	A	D	6	F	C	4	9
$S_3(x)$	2	E	F	5	C	1	9	A	B	4	6	8	0	7	3	D
$S_4(x)$	0	7	3	4	C	1	A	F	D	E	6	B	2	8	9	5

Linear Transform $L : \{0,1\}^{16} \rightarrow \{0,1\}^{16}$

$L(m) = m \oplus (m \lll 6) \oplus (m \lll 10)$

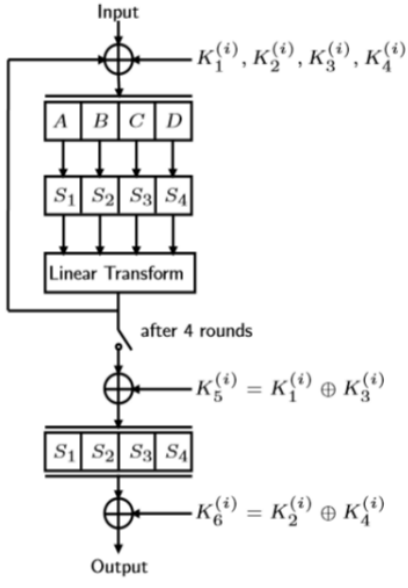

Figure 1.3: The structure of the block cipher in the Hummingbird Cryptographic Algorithm

Hummingbird is described as an ultra-lightweight cryptographic algorithm. It is widely believed to be the most capable cryptosystem for implementation in a resource-constrained environment. Hummingbird can achieve up to 147 and 4.7 times faster throughput for size-optimized and speed-optimized imple-

mentations respectively, when compared to the state-of-the-art ultra-lightweight block ciphers on similar platforms. Hummingbird is also designed to protect against the most common attacks such as birthday attacks, differential and linear cryptanalysis, structure attacks, algebraic attacks, cube attacks, etc. [2]. Moreover, extremely simple arithmetic and logic operations are extensively employed in Hummingbird for faster performance.

Message Length	Microcontroller Word Length [bit]	PRESENT [29] Encryption [ms]	Hummingbird Encryption [ms]	Performance Improvement
64-bit	8	2.40	3.38	-28.9%
	16	4.87	2.43	50.1%
128-bit	8	4.80	4.72	1.7%
	16	9.68	3.65	62.3%
192-bit	8	7.20	6.06	15.8%
	16	14.61	4.87	66.7%

Figure 1.4: Encryption performance comparison: Hummingbird vs. PRESENT

There are several emerging areas, such as automotive systems, sensor networks, healthcare, distributed control systems, the Internet of Things (IoT), cyber-physical systems, and the smart grid, in which highly constrained devices are interconnected, working in concert to accomplish some crucial task. Security and privacy can be VERY important in all of these areas. Applications often include direct interaction with the physical world. Consequently, a security incident might lead to asset damage or even personal injury and death. Hence, lightweight cryptographic algorithms should be designed very carefully keeping in mind high security standards. Security Analysis must be done against all fathomable attacks, like side channel attacks which are one of the most realistic threat against devices using lightweight cryptography.

1.3 Timing-based Side-channel attacks

In cryptography, a side-channel attack is any attack based on information gained from the physical implementation of a cryptosystem, rather than brute force or theoretical weaknesses in the algorithms. This information can be in the form of timing, power consumption, electromagnetic leaks or even sound. A timing-based side-channel attack is an attack based on measuring how much time various computations take to perform. In a timing-based attack, the attacker attempts to compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithms. Every logical operation in a computer takes time to execute, and the time can differ based on the input. With precise time measurements for each operation, an attacker can get information about the secret key embedded in the device.

Our main motivation upon taking this project was the fact that there is no evidence of a side-channel attack being analysed on the Hummingbird cipher till date, let alone timing-based. Only theoretical attacks such as classical cryptanalysis, birthday attacks etc. have been studied and the algorithm is found to be resistant against all of these attacks. While this is a good thing, we feel that the category of side-channel attacks should not be ignored bearing in mind the increasing security demands of the ever-changing world of pervasive computing.

Taking all of this into consideration, we try and mount a timing-based side-channel attack on the Hummingbird block cipher. A divide-and-conquer approach in view, we target only the first subround of the first block cipher(E_{k_1}) as shown in figure 1.5. The focus, like any other attack, was towards studying the execution time values and gleaning some information about the key. For all practical purposes, the 16-bit input to this round is referred to as ”plaintext” and the 16-bit output of this round is referred to as ”output” throughout the rest of the report, also the 16-bit key($K_1^{(i)}$) is referred to as ”key”.

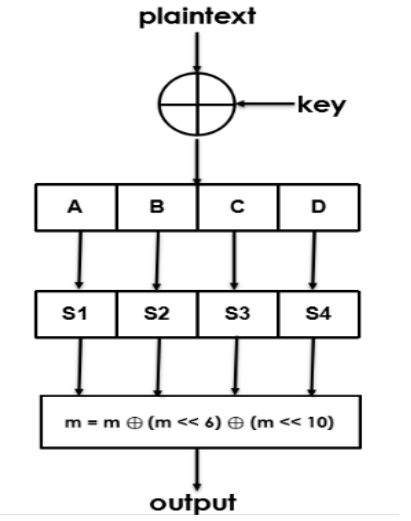

Figure 1.5: First subround-First block cipher:Hummingbird

2. Timing Analysis: Software Implementation

2.1 System Specifications

The Hummingbird cryptography algorithm was implemented on a machine with Intel(R) i5 1.70 GHz processor. The installed memory(RAM) was 4.00 GB and the system type was 64-bit with an x64 based processor. C++ 14 was used for implementation, the compiler being MingGW-W64.

2.2 Execution time analysis

Since a timing-based attack had never been attempted earlier, our first task was to find out whether mounting such an attack was possible on the Hummingbird block cipher or not. That is, whether execution time of the algorithm depended on the value of plaintext and key. To determine the existence of this dependency, a very simple experiment was performed. The first subround of the first block cipher (see Figure 1.5) was implemented in C++ 14. 10 sets of keys were selected for the same plaintext and execution time was recorded. C++ QueryPerformanceCounter functions were used to measure execution time as shown in Figure 2.1. A plot of execution time vs. key was obtained (Figure 2.2).

```
LARGE_INTEGER frequency;
QueryPerformanceFrequency(&frequency);
LARGE_INTEGER start;
QueryPerformanceCounter(&start);
unsigned short temp = in^(unsigned short)key;
string binary = bitset<16>(temp).to_string();
binary.replace(0,4,s1[stoi(binary.substr(0,4), NULL, 2)]);
binary.replace(4,4,s2[stoi(binary.substr(4,4), NULL, 2)]);
binary.replace(8,4,s3[stoi(binary.substr(8,4), NULL, 2)]);
binary.replace(12,4,s4[stoi(binary.substr(12,4), NULL, 2)]);
unsigned short temp2 = (unsigned short)stoi(binary, NULL, 2);
output = temp2 ^ rol(temp2,6) ^ rol(temp2,10);
LARGE_INTEGER end;
QueryPerformanceCounter(&end);
double timeDiff = (static_cast<double>(end.QuadPart- start.QuadPart) / frequency.QuadPart);
```

Figure 2.1: Code used for measuring execution time

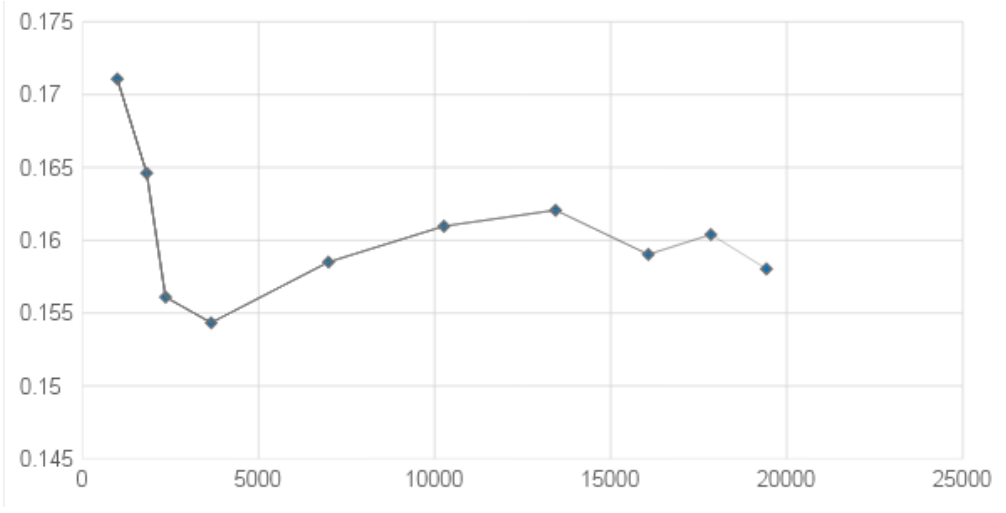

Figure 2.2: Plot of execution time(μs) vs. key $K_1^{(i)}$

Observing the graph, It was evident that execution time changed with change in key values. The next task was to study this relationship further and to design an attack. At this stage in our analysis, execution time seemed to be a good measure for studying the difference between key values.

To better study the interdependence of input values and execution time, various plots were observed. The first two plots (Figure 2.3 and 2.4) try to explore the relationship between Hamming Weight of key and execution time. These were obtained by measuring execution time for all the keys and then plotting them Hamming-Weightwise. Plaintext was kept constant. Our purpose here was to find out whether any Hamming Weight follows any special property wrt. Execution time. We were expecting a peak or depression in the graph corresponding

to such a Hamming Weight. This would have allowed us to predict the Hamming Weight of the key by measuring the execution time while launching our attack. The green and black lines here represent the maximum and minimum values while the blue line gives the average over each Hamming Weight.

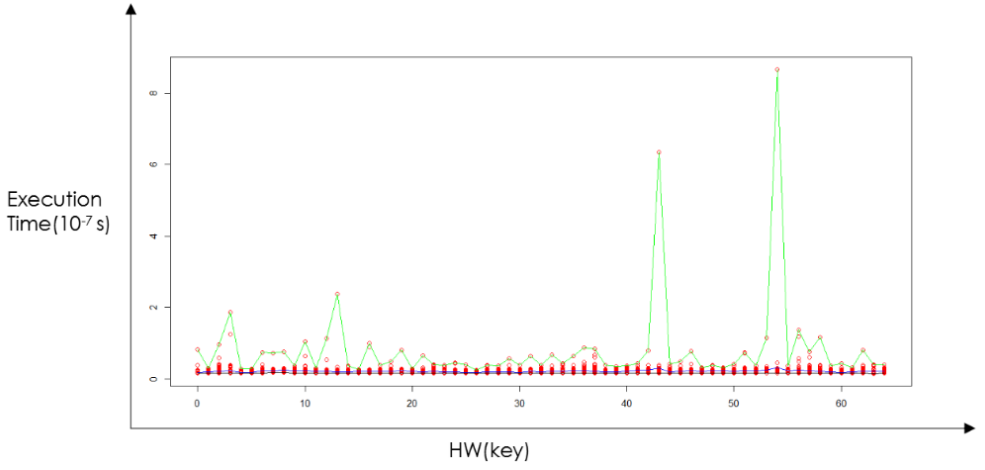

Figure 2.3: Execution time vs. Hamming Weight of key

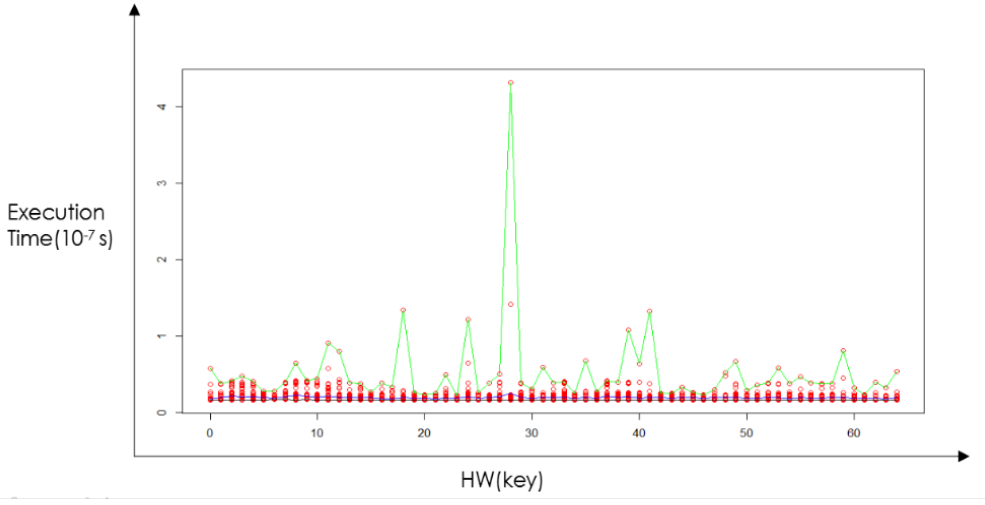

Figure 2.4: Execution time vs. Hamming Weight of key: 2nd run

$$ET_1(hw) = max(ET(x,k) \forall k \text{ s.t. } HW(k) = hw)$$
$$ET_2(hw) = min(ET(x,k) \forall k \text{ s.t. } HW(k) = hw)$$
$$ET_3(hw) = mean(ET(x,k) \ \forall k \text{ s.t. } HW(k) = hw)$$

here,

ET(x,k): execution time for plaintext = x and key = k

HW(k): Hamming Weight of key k

Looking at these graphs, two points were suddenly very clear. There is no Hamming Weight which obeys any special property with respect to execution time. Values were sufficiently random to safely conclude this. Moreover, the data obtained was very inconsistent. Multiple runs resulted in a totally different graph.

Yet another effort to extract some information from these graphs was made. This time, the mean and standard deviation of each Hamming Weight was recorded and the confidence interval $[\mu - \sigma, \mu + \sigma]$ was plotted. The purpose was to find out if these confidence intervals can distinguish between different Hamming Weights.

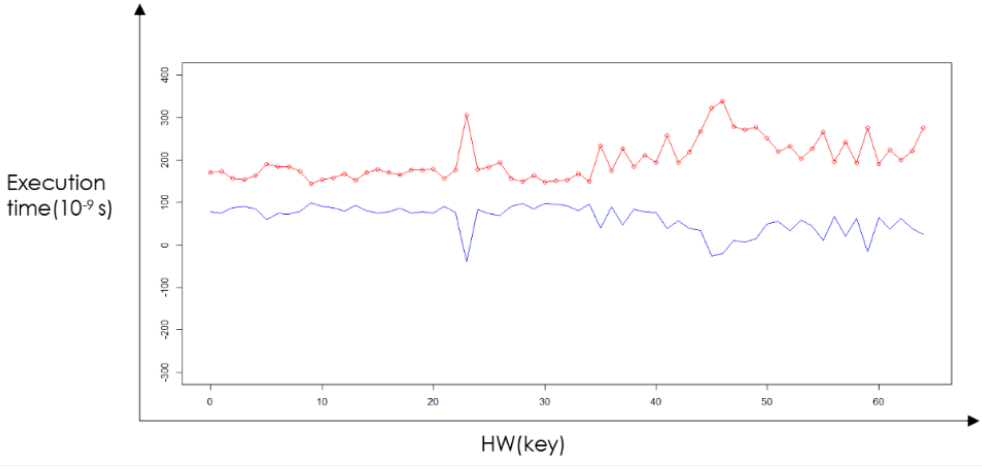

Figure 2.5: Confidence interval of execution time vs. Hamming Weight of key

$$ET_1(hw) = \mu - \sigma; ET_2(hw) = \mu + \sigma$$

where,

$\mu : mean(ET(x,k)\ \forall k \text{ s.t. } HW(k) = hw)$

$\sigma : stddev(ET(x,k)\ \forall k \text{ s.t. } HW(k) = hw)$

Had these intervals not overlapped, confidence interval of execution time could have served as a good distinguisher. But, as can be observed from figure 2.5, we were not so lucky.

The next task was to study the relationship between plaintext and execution time. Figure 2.6 illustrates this for Hamming Weight = 2. The graphs for each Hamming Weight were obtained by taking an average of all the key values of that particular Hamming Weight for the corresponding plaintext. 17 graphs (one for each Hamming Weight of key) were obtained in this manner.

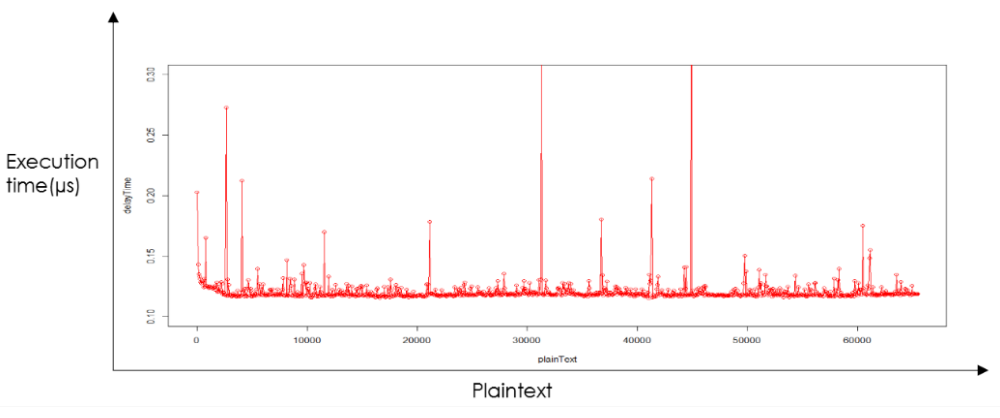

Figure 2.6: Execution time vs. plaintext : hw = 2

$ET_{hw}(x) = mean(ET(x,k)\ \forall k \text{ s.t. } HW(k) = hw)$

where,

x: plaintext; *k: key*

The purpose here was to find out whether the whole graph can serve as a distinguisher for that particular Hamming Weight. A small experiment was per-

formed to determine this. Obtaining these 17 graphs, we obtained a similar graph, this time for an unknown key. Then a direct match (least Euclidean distance) was applied between this graph and the graphs obtained previously to predict the Hamming Weight of this key (see figure 2.7). Accuracy of our prediction algorithm was obtained by repeating the experiment for 1000 unknown keys and determining the number of keys for which the predicted value was equal to the actual value of Hamming Weight.

Profiling
    for each hamming weight(hw) from 0 to 15
        get $ET_{hw}(x)$ (see figure 2.6)

Attack
    for each unknown key, $k_{unknown}$
        get $ET_{observed}(x)$
        $hw_{predicted}$ = hw such that $<ET_{observed}, ET_{hw}>$ is minimum

Figure 2.7: Hamming Weight Prediction

The above experiment produced disappointing results. Out of 1000 times, the algorithm gave the correct value of Hamming Weight only 16 times. By this time, it was realized that launching an attack using execution time measured just once for any plaintext-key pair cannot give desirable results due to its highly inconsistent values.

2.3 Weighted execution time analysis

Since execution time did not produce desirable results, it was decided to condense our field of view and study the nature of this execution time, this time focusing on a constant plaintext and key value. A probability distribution of execution time was obtained by measuring it 20,000 times for the same plaintext and key. Various probability distributions were obtained for different pairs of plaintexts and keys. Table 2.1 shows all plaintext and keys for which pairs were

made. Figure 2.8 shows the probability distribution for plaintext = 0 and key = 0.

Plaintext	Key
0	0
32768	12345
65536	54321
	65536

Table 2.1: Plaintext and key pairs for probability distribution

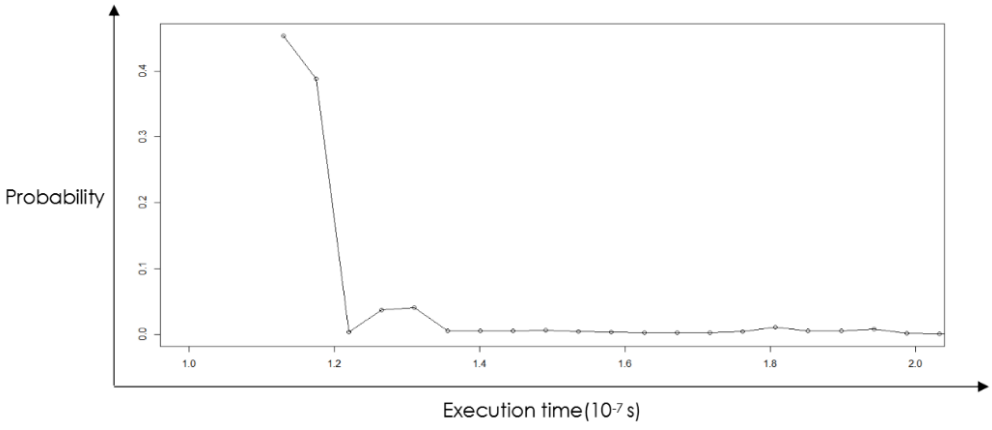

Figure 2.8: Probability distribution of execution time; plaintext=0 and key=0

Every probability distribution had the following salient features:

- Two values of execution time were observed almost 90% of the time.
- The highest occurring value could be observed with a probability of 0.5-0.6.
- The second highest occurring value could be observed with a probability of 0.3-0.4.
- These values were not necessarily the same for each plaintext and key pair.

To ensure more consistency, it was decided to replace the execution time with weighted execution time which was calculated as follows:

$$ET(x,k) = \frac{f1 * t1 + f2 * t2}{f1 + f2}$$

where,

f1: Frequency of execution time value occurring maximum number of times

t1: Execution time value occurring maximum number of times

f2: Frequency of execution time value occurring second maximum number of times

t2: Execution time value occurring second maximum number of times

All further experiments were done using this weighted execution time. Focus was shifted towards finding vulnerable values of keys which could follow certain criteria that would be detectable given plaintext and timing values. One method that immediately suggested itself was categorizing keys on the basis of the property plaintext = output. In an ideal cipher, there should not exist any pair of plaintext and key for which this occurs, even if the output is measured after just one round. Our next experiment was to find out whether this was true for Hummingbird. Every possible plaintext and key pair was taken and checked for the above property. Surprisingly, we found many values of plaintext and key for which this was true. A sample of the output is shown in figure 2.9. Dividing keys on the basis of the number of plaintexts that satisfy plaintext = output, 8 categories could be formed with the said number ranging from 0-7. Table 2.2 shows the number of keys in each category.

Key	PT0	PT1	PT2	PT3	PT4	PT5	PT6	PT7
0	60860							
1								
2	13282	49098	55674					
3								
4	47236							
5	19019							
6								
7								
8	52540							
9	27920	30528						
10								
11								
12	2585							
13	42135							
14	46295							
15	8174	25593	62964					

Figure 2.9: Key-plaintext pairs satisfying plaintext = output

Category	Number of keys
0	24397
1	23759
2	11965
3	4112
4	1055
5	199
6	47
7	2
Total	**65536**

Table 2.2: Number of keys for each category: plaintext = output

The above experiment gave us the idea of categorizing values of weighted execution time based on the Hamming Distance between plaintext and output. The next graph (Figure 2.10) was obtained by measuring the weighted execution time for each plaintext keeping the key constant and plotting it against the Hamming Distance between plaintext and output.

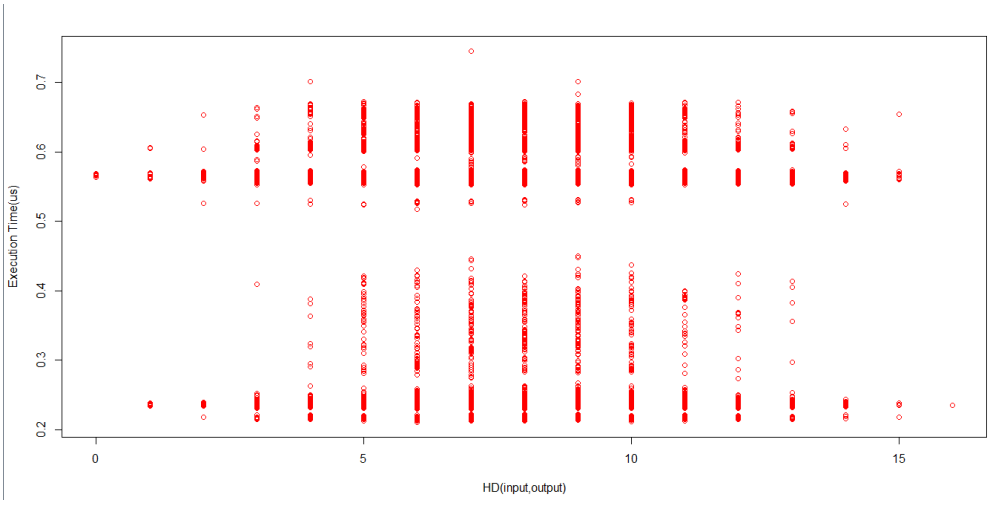

Figure 2.10: execution time(μs) vs. HD(plaintext,output); key = 2

Focusing specifically on the HD=0 and HD=1 data in the above graph, profiling and attack algorithms were designed. Since the actual key search space of 2^{16} was too big for analysis, a smaller search space, consisting of those keys for which number of plaintexts with HD = 0 is 6 (Table 2.2 Category 6) was taken. This was gradually increased to include values from categories 4, 5 and 7 as well. Changing either the profiling and attack algorithms or the size of the key search space, we performed 7 experiments (Figures 2.11 to 2.17). For all the experiments below, x stands for plaintext, k stands for key, $f_k(x)$ stands for output and ET stands for weighted execution time.

For every unknown key, our attack algorithms find a set of predicted key values. This set is referred to as the 'residual key space' and is obtained through elimination of keys which are less likely to be the actual keys. In such a case, there is a chance that the actual key is also eliminated from the residual key space. Keeping this in mind, the following parameters were used to measure the efficiency of these algorithms:

- Size of the residual key space (should be less)
- Number of times actual key is an element of the residual key space (should be more)

Experiment 1 (Figure 2.11) focuses on the data for HD = 0. For every key in the search space K, we find plaintexts that satisfy plaintext=output and get the range of weighted execution time for each key. Then for an unknown key, timing values are observed for all the plaintexts found previously. If any of the observed timing values for plaintexts of a particular key lies outside the prescribed range for that key (found by the profiling algorithm), the key is removed from the residual key space. Figure 2.11 describes the algorithm more precisely.

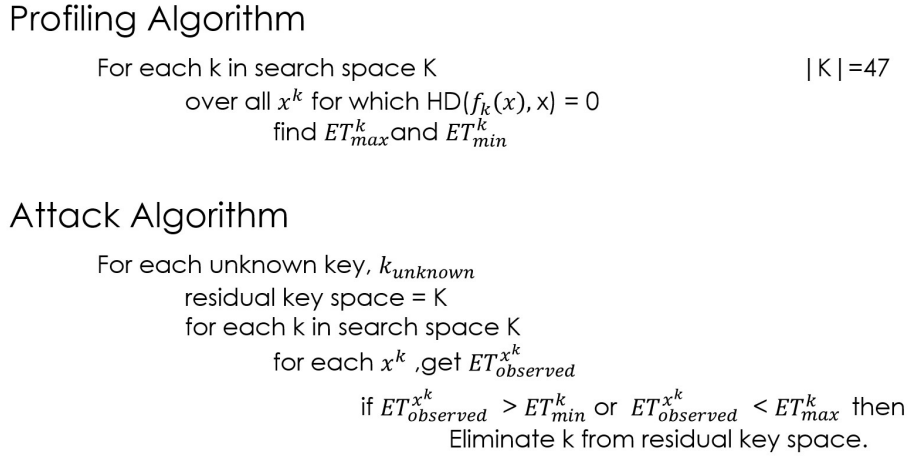

Figure 2.11: Weighted execution time- key prediction: Experiment 1

Experiment 2 (Figure 2.12) is similar to Experiment 1 except that instead of eliminating a key based on just one timing value, it is eliminated only if majority of timing values for plaintexts corresponding to that key lie outside the prescribed range.

Profiling Algorithm

For each k in search space K | K | =47
over all x^k for which HD($f_k(x)$,x) = 0
find $ET^k_{max}$and ET^k_{min}

Attack Algorithm

For each unknown key, $k_{unknown}$
residual key space = K
for each k in search space K
for each x^k ,get $ET^{x^k}_{observed}$
if for majority of x^k ,$ET^{x^k}_{observed}$ > ET^k_{min} or $ET^{x^k}_{observed}$ < ET^k_{max} then
Eliminate k from residual key space.

Figure 2.12: Weighted execution time- key prediction: Experiment 2

Experiment 3 (Figure 2.13) and 4 (Figure 2.14) are actually experiments 1 and 2 repeated for HD=1 data.

Profiling Algorithm

For each k in search space K | K | =47
over all x^k for which HD($f_k(x)$, x) = 1
find $ET^k_{max}$and ET^k_{min}

Attack Algorithm

For each unknown key, $k_{unknown}$
residual key space = K
for each k in search space K
for each x^k ,get $ET^{x^k}_{observed}$
if $ET^{x^k}_{observed}$ > ET^k_{min} or $ET^{x^k}_{observed}$ < ET^k_{max} then
Eliminate k from residual key space.

Figure 2.13: Weighted execution time- key prediction: Experiment 3

Profiling Algorithm

For each k in search space K | K | =47
over all x^k for which HD($f_k(x)$, x) = 1
find $ET^k_{max}$and ET^k_{min}

Attack Algorithm

For each unknown key, $k_{unknown}$
residual key space = K
for each k in search space K
for each x^k ,get $ET^{x^k}_{observed}$
if for majority of x^k ,$ET^{x^k}_{observed} > ET^k_{min}$ or $ET^{x^k}_{observed} < ET^k_{max}$ then
Eliminate k from residual key space.

Figure 2.14: Weighted execution time- key prediction: Experiment 4

Experiment 5 (Figure 2.15) reduces the plaintext set for which weighted execution time values for unknown key are required to be calculated. This time we just consider those values of plaintext which have timing values lying in the confidence interval $[\mu - \sigma, \mu + \sigma]$ of weighted execution time for each key.

Profiling Algorithm

For each k in search space K | K | =41
over all x^k for which HD($f_k(x)$, x) = 1
find ET^k_{max} ,ET^k_{min} , μ(ET^k), σ(ET^k)
select those x^k for which ET^k ϵ (μ(ET^k) - σ(ET^k) , μ(ET^k) + σ(ET^k))

Attack Algorithm

For each unknown key, $k_{unknown}$
residual key space = K
for each k in search space K
for each x^k ,get $ET^{x^k}_{observed}$
if $ET^{x^k}_{observed} > ET^k_{min}$ or $ET^{x^k}_{observed} < ET^k_{max}$ then
Eliminate k from residual key space.

Figure 2.15: Weighted execution time- key prediction: Experiment 5

Experiment 6 (Figure 2.16) repeats experiment 5 for HD=0 on a larger key set.

Profiling Algorithm

For each k in search space K, | K | =1303

over all x^k for which HD($f_k(x)$, x) = 0

find ET^k_{max} ,ET^k_{min} , μ(ET^k), σ(ET^k)

select those x^k for which ET^k ϵ (μ(ET^k) - σ(ET^k) , μ(ET^k) + σ(ET^k))

Attack Algorithm

For each unknown key, $k_{unknown}$

residual key space = K

for each k in search space K

for each x^k ,get $ET^{x^k}_{observed}$

if $ET^{x^k}_{observed}$ > ET^k_{min} or $ET^{x^k}_{observed}$ < ET^k_{max} then

Eliminate k from residual key space.

Figure 2.16: Weighted execution time- key prediction: Experiment 6

Experiment 7 (Figure 2.17) is an exact repetition of experiment 5 on a larger key set.

Profiling Algorithm

For each k in search space K | K | =1299

over all x^k for which HD($f_k(x)$, x) = 1

find ET^k_{max} ,ET^k_{min} , μ(ET^k), σ(ET^k)

select those x^k for which ET^k ϵ (μ(ET^k) - σ(ET^k) , μ(ET^k) + σ(ET^k))

Attack Algorithm

For each unknown key, $k_{unknown}$

residual key space = K

for each k in search space K

for each x^k ,get $ET^{x^k}_{observed}$

if $ET^{x^k}_{observed}$ > ET^k_{min} or $ET^{x^k}_{observed}$ < ET^k_{max} then

Eliminate k from residual key space.

Figure 2.17: Weighted execution time- key prediction: Experiment 7

Table 2.3 gives a summary of the results of all the above experiments, each being repeated over 100 unknown keys.

Experiment Number	Size of the residual key space	Number of times expected key $\in$ residual key space
1	18-34	59
2	24-35	65
3	14-29	54
4	26-39	75
5	26-39	77
6	624-737	56
7	1066-1163	73

Table 2.3: Results- Experiments 1 to 7

2.4 Miscellaneous experiments

This section describes a couple of miscellaneous experiments done on the Hummingbird block cipher software implementation. These experiments were performed during the course of the project. Some interesting observations were obtained but due to lack of resources and time, they could not be pondered upon any further.

2.4.1 Sensitivity analysis

The first of the aforementioned experiments was done with the purpose of observing the effect on execution time on flipping exactly one bit of the key. Plaintext-key pairs were taken and execution time was measured before and after flipping each and every bit one by one. Output for plaintext=0 and key=0 is shown in figure 2.18.

Bit flipped	Time before flipping	Time after flipping
0	0.205274	0.0543373
1	0.0482998	0.0482998
2	0.0482998	0.0422623
3	0.0422623	0.0482998
4	0.0422623	0.0422623
5	0.0422623	0.0422623
6	0.0422623	0.0422623
7	0.0422623	0.0482998
8	0.0422623	0.0422623
9	0.0422623	0.0422623
10	0.0362248	0.0482998
11	0.0422623	0.0422623
12	0.0422623	0.0422623
13	0.0724497	0.0422623
14	0.0422623	0.0482998
15	0.0422623	0.0482998

p=0011000000111001;k=0000000000000000

Bit flipped	Time before flipping	Time after flipping
0	0.289799	0.0905621
1	0.0965996	0.0905621
2	0.0905621	0.0905621
3	0.0845246	0.0845246
4	0.0845246	0.0905621
5	0.0845246	0.0905621
6	0.0845246	0.0905621
7	0.0845246	0.0905621
8	0.0845246	0.0905621
9	0.0845246	0.0905621
10	0.0845246	0.0905621
11	0.0784872	0.0905621
12	0.0845246	0.0905621
13	0.102637	0.0905621
14	0.0845246	0.0845246
15	0.0905621	0.0845246

p=0011000000111001;k=0011000000111001

Figure 2.18: Sensitivity analysis- sample output; plaintext =0 ,key = 0

It could be observed that the flipping of one bit had an effect on the execution time. However, no explicit relation with bit significance could be found. Moreover, our purpose was to propose an attack, and this experiment did not provide any future insight in that direction.

Still, we feel that the dependency between bit significance and execution time can be studied more thoroughly. One way that suggests itself is the repetition of the experiment with weighted execution time, which gives clearer results than execution time, as we established later. The cumulative effect of bit significance and Hamming Distance of 'key before flipping' and 'key after flipping' can be observed by carefully recording these results. Better and more dedicated systems can be used for a precise measurement of execution time.

2.4.2 Key vulnerability

The second experiment under this category was focused towards extracting vulnerable plaintext-key pairs for the first subround. This involved evaluation of certain linear combinations of plaintext and output bits. Taking a value of key k, outputs were recorded for all plaintexts. Then, for two 16-bit numbers a and b, the probability

$$Pr(\sum_{0..15}^{\oplus} a_i x_i \oplus \sum_{0..15}^{\oplus} b_i y_i = 0)$$

was observed. This was done for k=12345 and 100 random a and b values. Figure 2.19 shows a part of the output.

a	b	frequency
6704->0001101000110000	27307->0110101010101011	32826
27654->0110110000000110	22275->0101011100000011	32706
12944->0011001010010000	19789->0100110101001101	32814
11290->0010110000011010	7447->0001110100010111	32876
4772->0001001010100100	18019->0100011001100011	32688
25630->0110010000011110	15732->0011110101110100	32848
16783->0100000110001111	19925->0100110111010101	32740
17517->0100010001101101	7897->0001111011011001	33032
25515->0110001110101011	11494->0010110011100110	32660
3660->0000111001001100	26184->0110011001001000	33036
3964->0000111101111100	20316->0100111101011100	32708
12410->0011000001111010	29009->0111000101010001	32750
28386->0110111011100010	12256->0010111111100000	32500
4617->0001001000001001	17812->0100010110010100	32804
146->0000000010010010	25511->0110001110100111	32722
32619->0111111101101011	6530->0001100110000010	32636
3526->0000110111000110	2386->0000100101010010	32730
28112->0110110111010000	23139->0101101001100011	32778
10292->0010100000110100	7955->0001111100010011	32972
2224->0000100010110000	21405->0101001110011101	32608
17469->0100010000111101	1042->0000010000010010	32734
15779->0011110110100011	23347->0101101100110011	32840
17256->0100001101101000	10511->0010100100001111	32530

Figure 2.19: Linear combination probability; key=12345

Taking an average of the 100 probabilities so obtained, we get the value $Pr = 0.49992934$. For an ideal cipher, this value should be very close to 0.5. If for any key value, this probability deviates far from 0.5, that key could be regarded as vulnerable. In principle, to prove the existence of such vulnerable keys, this experiment should be repeated for all keys and all possible values of a and b. However, such a program would have a computational complexity of $O(2^{48})$, and our machines were simply not capable of running such a program in any reasonable amount of time.

The above analysis can be considered similar to the one done in linear cryptanalysis, although it is different for two reasons. First, the input and output do not correspond to that of the whole cipher; instead they correspond only to the first subround. Second, our objective towards finding vulnerable keys was to observe their timing behaviour as compared to other keys, as opposed to launching a theoretical attack.

Talking about the future scope of this experiment, we maintain that if probability values for all possible keys are somehow calculated, vulnerable keys can perhaps be found. Execution time comparisons for these keys as compared to other keys can be done thereafter. In fact, the relation between execution time values and probability can be explored. However, since it was not possible with the limited resources available to us, we suspended this analysis here.

3. Timing Analysis: Hardware Implementation

3.1 Implementation Details

The hardware implementation of the Hummingbird block cipher was done in VHDL. Xilinx ISE 14.7 and ISim simulator were used for implementation and simulation respectively. Post-Place and Route Simulation was done to obtain information about the time when output was obtained after plaintext and key had been set. For the rest of this chapter, we refer to this time as simulation time. For better recording of data, Test Benches were implemented in Verilog. The Verilog function $fmonitor() was used to record changes in output.
The device properties were as follows:
Family-Spartan6
Device-XC6SLX45T
Package-FGG484

The Device Utilization Summary from the Place and Route Report is shown in Figure 3.1.

```
Device Utilization Summary:

Slice Logic Utilization:
  Number of Slice Registers:                     0 out of  54,576    0%
  Number of Slice LUTs:                         29 out of  27,288    1%
    Number used as logic:                       29 out of  27,288    1%
      Number using O6 output only:              18
      Number using O5 output only:               0
      Number using O5 and O6:                   11
      Number used as ROM:                        0
    Number used as Memory:                       0 out of   6,408    0%

Slice Logic Distribution:
  Number of occupied Slices:                    15 out of   6,822    1%
  Number of MUXCYs used:                         0 out of  13,644    0%
  Number of LUT Flip Flop pairs used:           29
    Number with an unused Flip Flop:            29 out of      29  100%
    Number with an unused LUT:                   0 out of      29    0%
    Number of fully used LUT-FF pairs:           0 out of      29    0%
    Number of slice register sites lost
      to control set restrictions:               0 out of  54,576    0%
```

Figure 3.1: Device utilization summary- VHDL implementation

3.2 Analysis

Similar to software implementation, our first task here was to find out whether mounting a timing-based attack was possible. 10 values of keys were thus selected to verify this and the simulation time was measured for plaintext = 12451. Figure 3.2 shows the graph obtained.

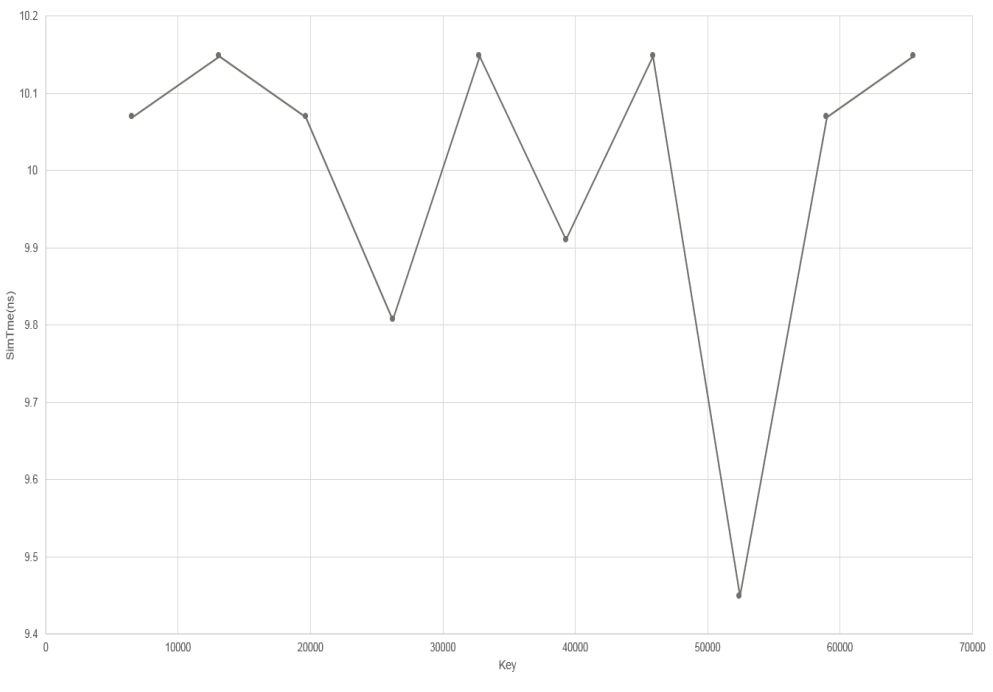

Figure 3.2: Simulation time vs. Key; plaintext = 12451

What remained now was to determine the parameters that simulation time depended upon. On multiple simulations, as long as device specifications are not changed, it was observed that simulation time depended upon three and only three things- **key**, **plaintext** and the **initial value of output** before key and plaintext are changed. Also, unlike execution time in the software implementation, the value of simulation time remained consistent i.e. it did not change with multiple runs.

An attempt at characterization of the simulation time on the basis of Hamming Weight of key, Hamming Weight of plaintext and the Hamming Distance between initial and final value of output was done. Since simulation time depended on the key, it was only natural to think that it would depend linearly on the Hamming Weight of the key. Similar speculations were made for plaintext and output. For output, it was considered that, the more the number of bits that need to be flipped (Hamming Distance), the greater the simulation time. To learn the truth about these speculations, the following graphs were plotted:

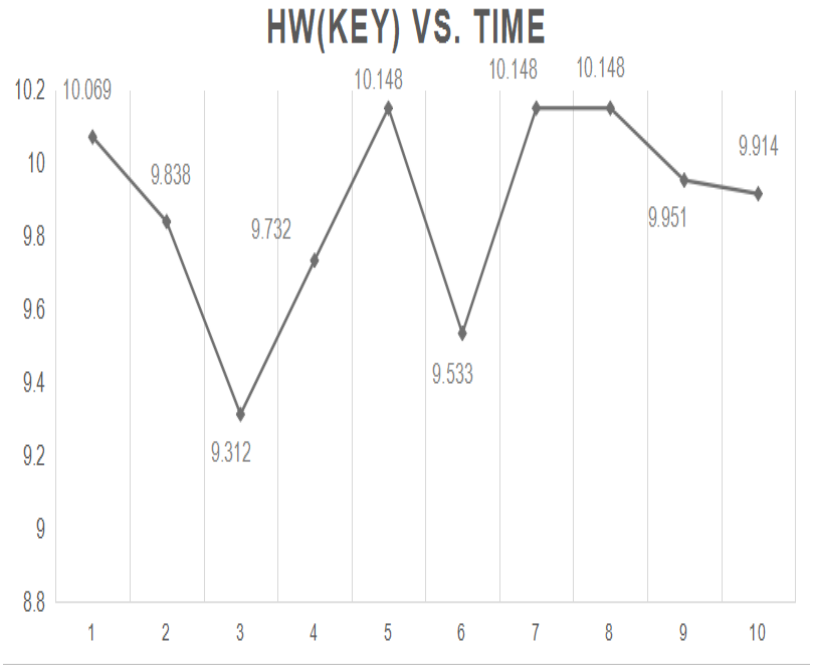

Figure 3.3: HW(key) vs. Simulation Time(ns)

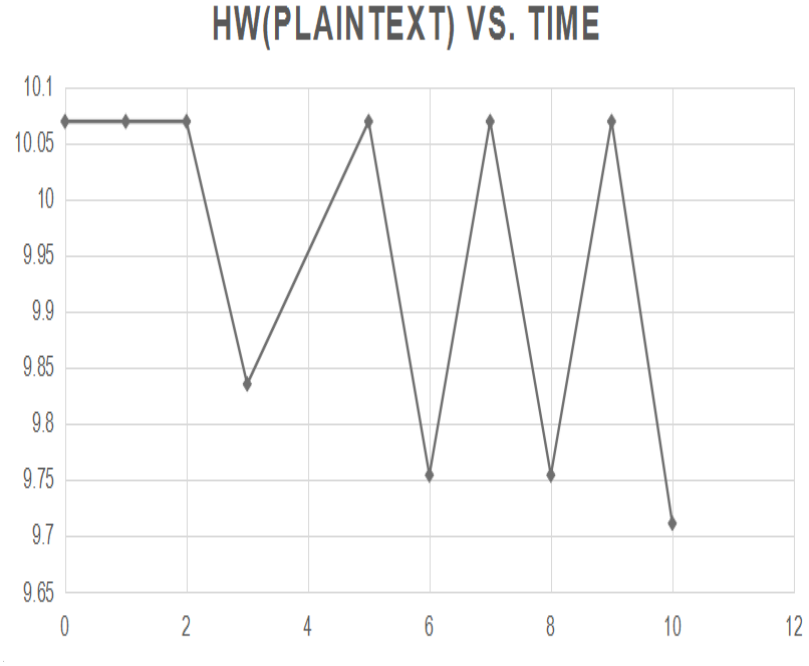

Figure 3.4: HW(plaintext) vs. Simulation Time(ns)

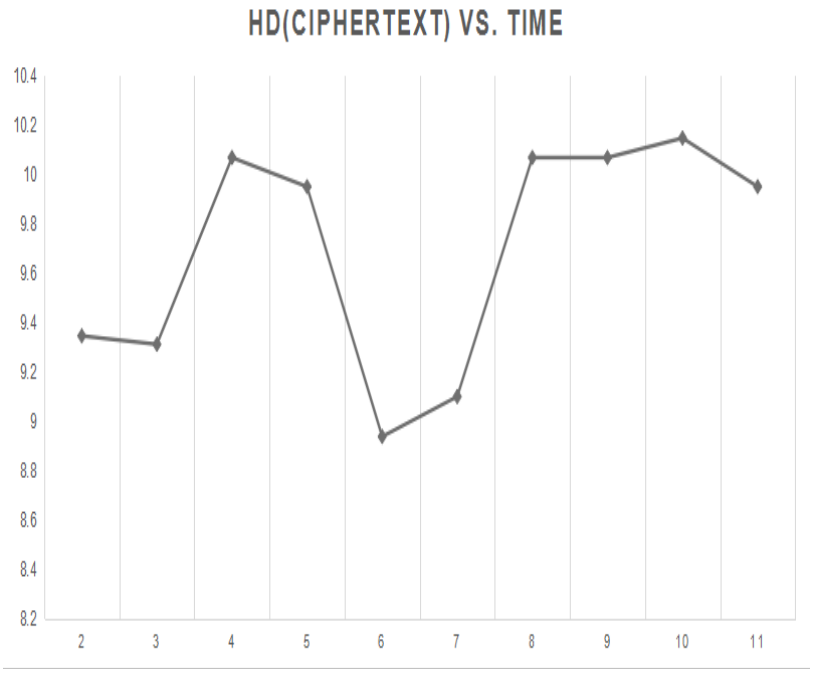

Figure 3.5: HD($output_{initial}$,$output_{final}$) vs. Simulation Time(ns)

While plotting the graph for one parameter, the other two parameters were kept constant. However, a linear relationship could still not be observed with any of the three parameters. Still another attempt at characterization was made. In the software implementation, we had collected information about weighted execution time keeping in mind the Hamming Distance between input and output (see Figure 2.10). A similar attempt was made here taking into consideration classes HD=0 and HD=1 specifically. The graph in Figure 3.6 has been obtained by taking categories 4, 5, 6 and 7 of keys from Table 2.2, getting their corresponding plaintexts which satisfy HD(plaintext,output)=0 and HD(plaintext,output)=1 and plotting simulation time for each pair vs. key. Our purpose was to determine whether the ranges of simulation time for HD=0(yellow points) and HD=1(blue points) were different. This would have meant that simulation time can be characterized on the basis of Hamming Distance between plaintext and output.

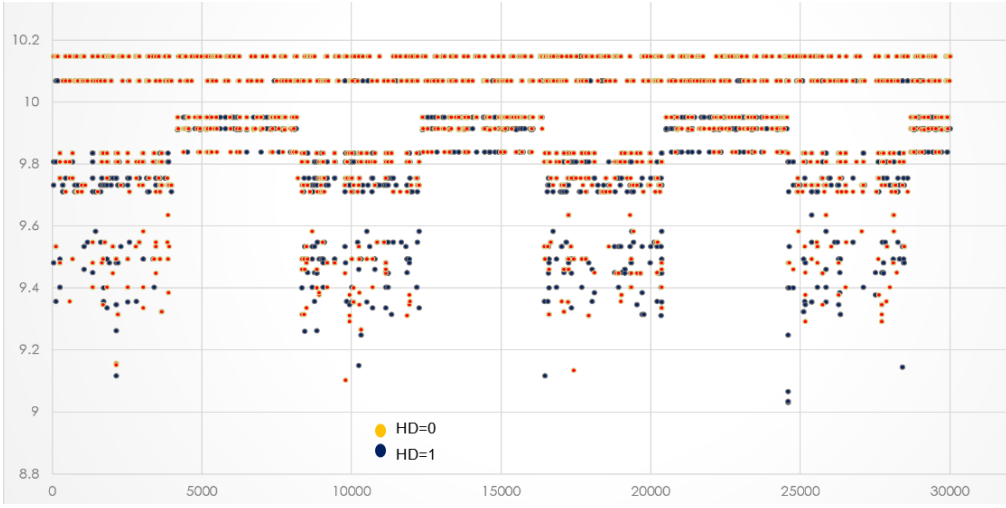

Figure 3.6: Simulation time(ns) vs. Key; HD=0 and HD=1

It can be seen from the above graph that ranges of values for HD=0 and HD=1 are not distinct. Hence, even this attempt at characterization was not fruitful. Still, it was known that simulation times are different for different keys. Even though the nature of this dependency was not clear, an attack could still be launched. We took a set of 10 plaintexts and recorded the simulation time for each key. Then, similar to the attack algorithms of Figures 2.11-2.17, a residual key space was defined for every unknown key. 10 simulation times were recorded corresponding to each plaintext and only those keys were kept in the residual key space for which these simulation times were the same as those recorded earlier. Figure 3.7 describes these profiling and attack algorithms more formally. Here, p stands for plaintext, k stands for key and ST stands for simulation time.

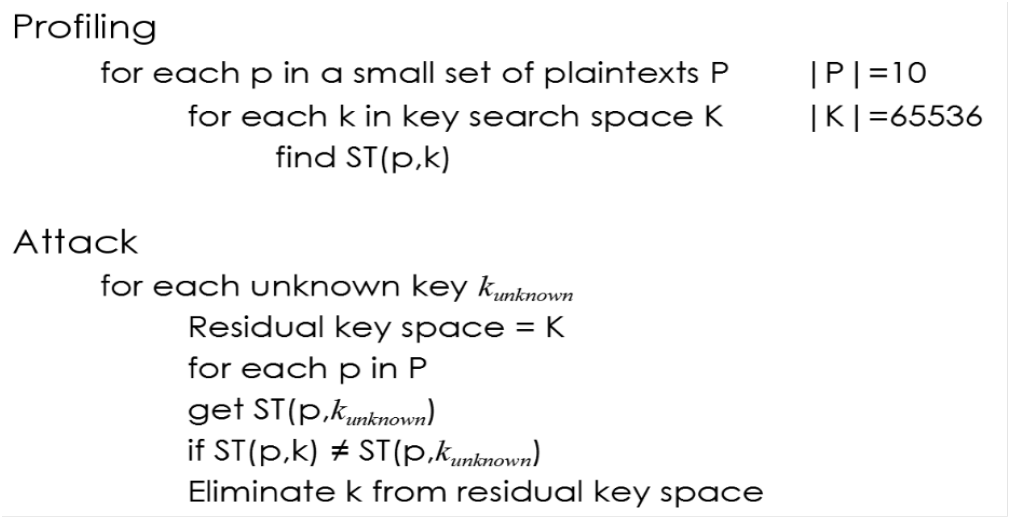

Figure 3.7: Simulation time- profiling & attack algorithms

To measure the efficiency of this algorithm, it was sufficient to determine the size of the residual key space. Unlike in the software implementation, here there wasn't a chance of the actual key being eliminated as simulation value did not change if plaintext, key and initial value of output were kept constant. Hence, the actual key was always a part of the residual key space. After repeating our experiment for 100 unknown keys,

Size range of the residual key space: 6 - 8192

Average size of the residual key space: 1771

We can reduce the size of the residual key space by increasing the size of the set of plaintexts P. Although the 10 plaintexts that we did our experiment with were chosen at random, efforts can be made towards finding the most efficient plaintext set P that reduces the size of the residual key space to a bare minimum. This can be done by better profiling. Due to time constraints, we could design only a very basic attack algorithm and did not work on making it more efficient. Time and space optimization can be applied on both the algorithms. Another observation that could not be worked upon was that some bits were being set later than others. A bit-wise analysis to separate such bits can be done.

4. Achievements and Results

Throughout the course of this project, our purpose was to analyze the dependence between the execution time of the first subround of the Hummingbird block cipher and the input provided to it. This was done to find possible glitches in implementation which could then help us gain information about the key. The following sections summarize our achievements and results in this regard.

4.1 Software Implementation

A thorough analysis of the change in execution time for the first subround of the Hummingbird block cipher with Hamming Weight of key and plaintext was completed, leading us to successfully establish the fact that execution time alone is not a good measure for mounting a timing-based attack. This fact was not established by a mere observation of the data; an attack algorithm which predicted the Hamming Weight of the key based on collected data was also tried and only after finding that it had a ridiculously low accuracy, execution time was rejected as a good parameter.

Analysis was not given up once it was found that weighted execution time can serve as a better parameter for launching an attack than execution time. Further, it was discovered that the subround does not follow ideal behaviour; there are some plaintext-key pairs for which plaintext and output are the same. Vulnerable keys were isolated based on the number of plaintexts which satisfied this property.

Profiling and attack algorithms with weighted execution time and Hamming

Distance of plaintext and output being the information were successfully designed. A measure of their efficiency was effectively established and the said efficiency correctly measured over a set of experiments. Results were obtained not only on the basis of the method of profiling and attack but also on the size of the key search space.

Result: On a reduced key search space, the proposed algorithms were able to reduce the number of possible keys by roughly half. On increasing the size of the key search space, this ratio reduced to approximately 0.2. The actual key was a part of the reduced key space about 70% of the time in both cases.

4.2 Hardware Implementation

A thorough recording of the variations in simulation time with respect to key was completed. The fact that simulation time depends only on key, plaintext and initial value of output, provided device specifications are not changed, was established. Further, it was also discovered that the relationship between simulation time and Hamming Weight of key, Hamming Weight of plaintext and Hamming Distance between initial and final value of output is not linear. Categorization of simulation time on the basis of Hamming Distance between plaintext and output fails.
A key prediction algorithm was successfully designed and its efficiency was measured fixing the value of one of the algorithm parameters i.e. size of the plaintext set P (see figure 3.7) for the experiment.

Result: The proposed algorithm was able to successfully reduce the entire key search space to 3% of its original size.

5. Conclusion and Scope of Future Work

The present study can be made to suggest that timing-based side-channel attack algorithms can indeed be launched on the Hummingbird block cipher. This was concluded from a series of observations which always proved that execution time/simulation time changes with change in key value. However, our conclusions are in no way absolute. As could also be noted at various points in the report, there is a lot of scope for future analysis. The rest of this chapter discusses our inferences and provides some ideas on how they can be built upon in the future.

5.1 Software implementation: Conclusions & Scope

Execution time is not a good measure for mounting a timing-based attack. This is because execution time data changes drastically with multiple runs. Aside from suggesting that it is not good for mounting a timing-based side-channel attack, this also suggests that apart from key and plaintext, timing values also depend upon other system variables like processor and memory utilization, cache hits and misses, throughput etc., the values of which may change phenomenally during the course of multiple runs. To account for these values, either a more dedicated system can be used for measurement of time or their relationship with execution time can be carefully studied.

Vulnerable keys can be identified on the basis of the number of plaintexts that satisfy the relation plaintext=output. Based on this property, we were

able to segregate keys into 8 categories (Table 2.2). A more rigorous analysis than the one presented in this report can be done for finding relations between execution time and categories. More ways of identifying vulnerable keys can be found.

Attack methods based on weighted execution time can reduce the key space by half and give a success rate of roughly 70%. This was established through a series of seven experiments (Figures 2.11-2.17). Future work can involve improvement of the proposed algorithms to get a success rate of nearly 100%. One way of doing this can be to carefully study the weighted execution time and determine the factors that it depends upon.

5.2 Hardware implementation: Conclusions & Scope

For one particular device, simulation time depends only on plaintext, key and the initial value of output. Sufficient number of simulations were done to reach this conclusion, although, exactly how it depends on key, plaintext and the initial value of output could never be determined. Efforts can be made in this direction.

Attack algorithms based on simulation time can reduce the key space to approximately 3%. This was when we collect data for 10 plaintexts while profiling (See figure 3.7). The efficiency can be improved by increasing the number of plaintexts. Efforts can be made towards finding the set of plaintexts which gives the best results.

We conclude this report with the hope that our work aids in valuable research targeted towards the betterment of humanity, even if it does so in a very small way. It was definitely a great learning experience and saying that we enjoyed every minute of it would not be an overstatement.

References

[1] Nigel Smart. *Cryptography: An introduction (3rd edition)*. McGraw-Hill College.

[2] Engels D., Saarinen MJ.O., Schweitzer P., Smith E.M. (2012) "The Hummingbird-2 Lightweight Authenticated Encryption Algorithm." In: Juels A., Paar C. (eds) *RFID. Security and Privacy. RFIDSec 2011.* Lecture Notes in Computer Science, vol 7055. Springer, Berlin, Heidelberg

[3] X. Chen, Y. Zhu, Z. Gong and Y. Luo, "Cryptanalysis of the Lightweight Block Cipher Hummingbird-1," *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies*, Xi'an, 2013, pp. 515-518.

[4] S. Saha, M. R. Islam, H. Rahman, M. Hassan and A. B. M. A. Hossain, "Design and implementation of block cipher in hummingbird algorithm over FPGA," *Fifth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, Hefei, 2014, pp. 1-5.

[5] Nicky Mouha. "The Design Space of Lightweight Cryptography". *NIST Lightweight Cryptography Workshop 2015*, Jul 2015, Gaithersburg, United States.

[6] B. Coppens, I. Verbauwhede, K. D. Bosschere and B. D. Sutter, "Practical Mitigations for Timing-Based Side-Channel Attacks on Modern x86 Processors," *2009 30th IEEE Symposium on Security and Privacy*, Berkeley, CA, 2009, pp. 45-60.

[7] Sadanandan, Sandeep and Mahalingam, Rajyalakshmi, ”Light Weight Cryptography and Applications”, *Novel Algorithms and Techniques In Telecommunications, Automation and Industrial Electronics 2008*, Springer, Netherlands.

[8] Lightweight Cryptography,
`https://www.cryptolux.org/index.php/Lightweight_Cryptography`

[9] Side-channel attacks,
`http://www.techdesignforums.com/practice/guides/side-channel-analysis-attacks/`

[10] What is RFID?,
`https://www.epc-rfid.info/rfid`