2015

Hardware Trojan Detection in Cryptography IP Cores by Library Encoding Method

Dinesh Varma Penumetcha
Wright State University

Follow this and additional works at: http://corescholar.libraries.wright.edu/etd_all

Part of the Electrical and Computer Engineering Commons

Repository Citation

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu.
Hardware Trojan detection in cryptography IP cores by library encoding method

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering

By

Dinesh Varma Penumetcha
B.E., Acharya Nagarjuna University, 2012

2015
Wright State University
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Dinesh Varma, Penumetcha ENTITLED **Hardware Trojan detection in cryptography IP cores by library encoding method** BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Engineering

Saiyu Ren, Ph.D.
Thesis Director

Brian D. Rigling, Ph.D.,
Department Chair
Department of Electrical Engineering
College of Engineering and Computer Science

Committee on Final Examination

Saiyu Ren, Ph.D.

Raymond E. Siferd, Ph.D.

Jiafeng. Xie, Ph.D.

Robert E. W. Fyffe, Ph.D.
Vice President of Research and Dean of the Graduate School
Abstract


Security is the primary issue in current internet world through both software and hardware. The ever increase in demand of consumer electronics requires less design cycle. To speedup design cycle, companies are approaching third parties for common applications IPs like USB, cryptography, DSP etc. These third parties can introduce a malicious content, which is called Trojan. Trojan in the netlist can activate only with special input/trigger. Available Trojan detection techniques like delay, area, power fingerprinting techniques and Automatic Test Pattern Generator (ATPG) method are not suitable as they take more time, less accurate. This thesis presents a hardware Trojan detection in cryptography IP cores by library encoding method. The final netlist of cryptography IP cores are encoded and decoded by using a script written in python to protect the design from Trojan insertion. This method of encoding and decoding detects even 0.0008% of Trojan area and disable the Trojan from activation.
# TABLE OF CONTENTS

1. INTRODUCTION .................................................................................................................. 1  
   1.1. Hardware Trojan Classification ................................................................................. 4  
   1.2. Trojan Detection Techniques ..................................................................................... 6  
   1.3. Motivation .................................................................................................................. 9  
   1.4. Thesis Organization ................................................................................................... 10  

2. ADVANCED ENCRYPTION STANDARD ............................................................................. 11  
   2.1. AES 128 Overview ..................................................................................................... 12  
   2.2. Rijndael algorithm basic components ................................................................. 14  
   2.3. Rijndael Transformations ......................................................................................... 16  
      2.3.1. Byte substitution: ............................................................................................. 16  
      2.3.2. Shift rows ......................................................................................................... 17  
      2.3.3. Mix columns .................................................................................................... 18  
      2.3.4. Key expansion .................................................................................................. 20  
   2.4. AES 128 Design Implementation ............................................................................ 23  
   2.5. Hardware Implementation ....................................................................................... 25  
      2.5.1. FPGA ................................................................................................................. 25  
      2.5.2. ASIC .................................................................................................................. 28  

3. INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA) ........................................ 31  
   3.1. IDEA Algorithm ....................................................................................................... 31  
   3.2. IDEA Encryption Components ............................................................................... 32
LIST OF FIGURES

Figure 1.1 Trojan insertion levels ........................................................................................................... 2
Figure 1.2 Hardware Trojan classification ................................................................................................. 4
Figure 1.3 Trojan (Source MERO rajat subra chakraborthy et.al [8]) .................................................... 5
Figure 1.4 Simple XOR gate ..................................................................................................................... 5
Figure 1.5 Hardware Trojans detection classification .............................................................................. 6
Figure 2.1 4*4 array representation of input 128 bit data ..................................................................... 14
Figure 2.2 AES Encryption Algorithm (Source: [17] William stallings pg 154 ) ............................... 15
Figure 2.3 Substitution Box array ........................................................................................................... 17
Figure 2.4 Shift rows of an array ............................................................................................................. 18
Figure 2.5 Conversion of 8 bit keys to 32 bit words ................................................................................ 20
Figure 2.6 Components in a round .......................................................................................................... 22
Figure 2.7 Conversion of 32 bit vector to 4*4 array ............................................................................... 23
Figure 2.8 Top level design of AES 128 Encryption ............................................................................. 23
Figure 2.9 Block level representation of byte substitution ................................................................... 24
Figure 2.10 Block level representation of key expansion ..................................................................... 24
Figure 2.11 Design interface with Chipscope cores ............................................................................... 25
Figure 2.12 FPGA result of AES encryption and decryption ................................................................. 28
Figure 2.13 simulation result AES encryption ....................................................................................... 29
Figure 2.14 simulation result of AES decryption .................................................................................. 30
Figure 3.1 key expansion .......................................................................................................................... 32
Figure 3.2 IDEA Internal implementation .............................................................................................. 34
Figure 3.3 Top Level design of IDEA Encryption .................................................................................. 36
Figure 3.4 IDEA key expansion design block diagram ....................................................................... 37
Figure 3.5 Simulation result of IDEA encryption .................................................................................. 38
Figure 3.6 Simulation result of IDEA decryption .................................................................................. 39
Figure 4.1 Netlist encoding ...................................................................................................................... 42
Figure 4.2 Netlist decoding ...................................................................................................................... 43
Figure 4.3 Trojan Insertion to the encoded netlist file ............................................................................ 43
Figure 4.4 AES Encryption with mux payload ................................................................. 45
Figure 4.5 IDEA Encryption with mux payload ............................................................... 46
Figure 4.6 Golden IP IDEA encryption synthesis result .................................................... 48
Figure 4.7 Trojan inserted IDEA encryption synthesis result .......................................... 49
Figure 4.8 library encoding design of IDEA encryption synthesis result ......................... 50
Figure 4.9 Golden IP IDEA decryption synthesis result .................................................. 52
Figure 4.10 Trojan inserted IDEA decryption synthesis result ....................................... 53
Figure 4.11 library encoding design of IDEA decryption synthesis result ....................... 54
Figure 4.12 Golden IP AES encryption synthesis result .................................................. 55
Figure 4.13 Trojan inserted AES encryption synthesis result ........................................ 57
Figure 4.14 library encoding design of AES encryption synthesis result ....................... 58
Figure 4.15 Golden IP AES decryption synthesis result .................................................. 59
Figure 4.16 Trojan inserted AES decryption synthesis result ....................................... 60
Figure 4.17 Library encoded AES decryption synthesis result ....................................... 61
Figure 5.1 GUI application ............................................................................................ 66
LIST OF TABLES

Table 2.1 Key length, Block size and Number of rounds for AES 128, 192, 256 .................. 13
Table 2.2 Standard Matrix ........................................................................................................... 18
Table 2.3 Data Matrix ..................................................................................................................... 18
Table 2.4 final array ....................................................................................................................... 20
Table 2.5 AES encryption Virtex6 board Results ............................................................................ 26
Table 2.6 AES encryption Virtex6 board Results ............................................................................ 27
Table 3.1 Sub keys to Encryption Rounds ....................................................................................... 33
Table 3.2 Sub keys to Decryption ................................................................................................... 36
Table 3.3 IDEA encryption Virtex6 board Results ........................................................................... 39
Table 3.4 IDEA decryption Virtex6 board Results ......................................................................... 40
Table 4.1 IDEA encryption golden IP area result .......................................................................... 47
Table 4.2 IDEA encryption golden IP power result ....................................................................... 47
Table 4.3 IDEA encryption Trojan inserted design area result ....................................................... 49
Table 4.4 IDEA encryption Trojan inserted design power result ................................................... 49
Table 4.5 IDEA decryption golden IP area result .......................................................................... 52
Table 4.6 IDEA decryption golden IP power result ....................................................................... 52
Table 4.7 IDEA decryption Trojan inserted design area result ....................................................... 53
Table 4.8 IDEA decryption Trojan inserted design power result ................................................... 53
Table 4.9 AES encryption golden IP area result ............................................................................ 55
Table 4.10 AES encryption golden IP power result ...................................................................... 55
Table 4.11 AES encryption Trojan inserted design area result ....................................................... 56
Table 4.12 AES encryption Trojan inserted design power result ................................................... 56
Table 4.13 AES decryption golden IP area result ............................................................................ 59
Table 4.14 AES decryption golden IP power result ...................................................................... 59
Table 4.15 AES decryption Trojan inserted design area result ....................................................... 60
Table 4.16 AES decryption Trojan inserted design power result ................................................... 60
Table 4.17 Results of IDEA, AES encryption an decryption IP cores ............................................. 63
ACKNOWLEDGEMENT

My journey at Wright State University (WSU) is meaningless without all these people and I will always cherish forever for all the experiences that I had at WSU.

I would like to express the deepest appreciation to my committee chair, Dr. Saiyu Ren for her attitude towards research and teaching. Without her continuous support and help this thesis would not have been possible. I would also like to thank Dr. Raymond E. Siferd and Dr. Jiafeng Xie for being on my defense committee.

I am sincerely thankful to my Academic advisor Dr. Ryan Hamilton and system administrator Mike VanHorn for their support throughout my Master degree program at Wright State University. I would like to express my sincere gratitude to the Director of International Collaborations and Graduate Programs, Ms. Swapna Nair for giving me a wonderful opportunity to work with her team and her inputs in problem solving always amazed me.

My special thanks to the World’s greatest and my lovely mother Vani for providing best in everything. Mom, I always admire you for the way you look at life. I would like to thank my brother Trinadh and sister Neeraja for their encouragement. I would also like to thank my roommates, friends for making my stay at wright state university as one of the most memorable mark in my life. Last but not least, thanks to my dearest friend who helped in reviewing this report.
1. INTRODUCTION

In the era of connected world where machine to machine communication significantly evolves a new market called Internet of Things (IoT). As per latest technology reports, there will be more than 50 billion [1] devices to be connected over internet in next 5 years. So the major market share will go to the world of semiconductor companies which design Integrated Circuits (IC’s). Though IoT will make the environment around us smarter, security is a primary concern. Also with the new demand in IC market and globalization of foundries; there raises a question of security. These questions about security result in evolving the new term called hardware cryptography, where the designs can either be in system level, Intellectual Property (IP) level, RTL level, gate level design or layout level designs securing from Hardware Trojans [2].

Hardware Trojan is an undesired design in circuit which alters the functionality or destroys the complete circuit. With the intrusion of hardware Trojans into the world of integrated circuits, which are backbone for all electronic devices from mobiles to defense applications, securing IC’s becomes a major problem. These Hardware Trojans inserted into final IC’s like Application Specific Integrated Circuits (ASIC) or reconfigurable devices like Field Programmable Gate Array (FPGA) make IC’s malfunction or steal information potentially. Due to these kind of threats, defense agencies like DARPA initiated research in those areas to secure the IC in applications, where security is most important and named it to be “Trust in IC’s“.
With the increase in complexity of systems and a need for common functionalities like audio or video encoders and decoders, there evolves a new term called Intellectual Property (IP) cores. Many IP vendor companies simply design these IP cores at Register Transfer Level (RTL) level and sell them to customers. These third party IP cores are more vulnerable [3] to Trojans due to easy netlist access of designs. On an average, one system on a chip (SoC) may contain more than ten IP cores, increasing the difficulty in
detection of Trojans by multifold. As described above, Hardware Trojans can be inserted into different levels of chip design. Fig. 1.1 shows the hierarchies of chip design, Trojan insertion possibilities and its effects.

System on chip level: This stage functions as collection of different IP cores. In this level different IP cores communicate with each other based on the functionality requirements. So, the Trojan insertion can disrupt the protocols of the design or change the entire functionality of system.

IP core level: IP cores are the reusable blocks of logic design used in front end digital IC design. The most common IP cores are Digital Signal Processing (DSP) cores, cryptographic cores or error correction cores. This level of Trojan insertion causes damage to the functionality of IP cores like bypassing outputs or altering the outputs when certain conditions met etc. [4].

CAD Tool level: Electronic Design Automation (EDA) companies like Cadence, Mentor Graphics, Synopsys and many more are responsible for developing tools which are widely used in almost all stages of chip design. These tools are automated with the help of scripting which runs in the background to perform either synthesis or simulation. Trojan insertion at this level might be very low, as companies developing the tools take necessary steps ensuing high level of security to the customers [5].

Behavioral Level: Behavioral means representation of the circuit design in one of the available hardware description languages (HDL) like VHDL or Verilog. Once design is described in HDL, then internally it will be realized as the data and control paths describe circuit behavior. Trojans mostly attack the signals with low observability [6].
**RTL Level:** At this level, designs are represented in registers. Even a small malicious logic is difficult to detect during standard tests. But research shows that rouge designer has large flexibility to insert any kind of Trojan [7].

**Gate or Netlist Level:** Designs represented in RTL level are converted to netlist level using available synthesis tools with input as a RTL design file and standard cell library. In this level, Trojans can either be in standard cells or in netlist file. Detecting Trojans at this level is a bit difficult.

**Transistor level:** Trojans inserting at this level may change the transistor width resulting in changing power and timing characteristics also varying the device characteristics.

**Layout level:** Trojans may change the width of metal layers, size of wires.

### 1.1. Hardware Trojan Classification

Trojans are classified based on behavior, activation and action. Each of them is discussed below in detailed.

![Figure 1.2 Hardware Trojan classification](image-url)
**Behavior:** Malicious circuit design inserted can be subdivided based on the type of design implementation. It can either be combinational design or sequential designs. Firstly, combinational designs are implemented only with Boolean logic gates as shown in Fig. 1.3. Here the Trojan circuit is only activated when \(a=0\), \(b=1\) and \(c=1\), otherwise the circuit operates without any malfunction. Once a special test pattern enables the Trojan logic, output of \(S'\) is altered. On the other hand, sequential designs contain memory elements which is on the right side of Fig. 1.3, where counter value is incremented when \(a=1\), \(b=0\) at rising edge of clock. Once counter reaches a specific value the functionality of \(S'\) is altered.

![Figure 1.3 Trojan (Source MERO rajat subra chakraborthy et.al [8])](image)

**Activation:** Combinational and sequential designs are only activated with some special patterns. Based upon method of activation, Trojans are classified as Always-On, Internally activated and externally activated.

Always-On Trojans do not need any special inputs to be triggered i.e. in always-on type a simple ex-or gate can be inserted. An internal signal is connected to one of the inputs and logic ‘1’ to another. The result of this type of Trojan is a complement of the internal signal.

![Figure 1.4 Simple XOR gate](image)
Internally activated Trojans are not activated till specific condition is met i.e. Trojans can be activated only with special patterns. Activated Trojan modify the functionality of designs. In externally activated, the adversary drives an external signal to activate Trojans [9].

**Action:** Activated Trojan can either bypass key information of user, named as transmit information, alter function or specifications. Author in [9] describes a *modify function*, Trojan alters chip function by adding an extra logic or destroying entire chip, whereas *modify specification* Trojans alter properties like delay by modifying transistor parameters. *Transmit information* type Trojans may compromise sensitive information either by radio emissions or through covert channels built at the output of altered circuit.

### 1.2. Trojan Detection Techniques

![Hardware Trojans Detection Diagram](image)

**Figure 1.5 Hardware Trojans detection classification**
Lot of research has been going on in the area of Trojan detections. Till now, there is no single method detecting all the existing Trojans. Trojans can be detected at different levels of design, and the techniques depend on chip manufacturing steps. This level of Trojan detection technique is subdivided into post fabrication and pre fabrication.

**Post fabrication detection technique**

If a team of designers and testers are unable to detect Trojans in the front end fabrication of the chip design, then it should be detected in post fabrication before it is released to consumers. Post fabrication techniques are subdivided into physical testing and monitoring parameters like power, delays on chip.

**Physical testing**

Scanning Optical Microscope (SOM) method scans IC’s by projecting narrow laser beam on to chips revealing circuit functionality at the transistor level. This method can also be named as forensic analysis of microelectronic circuits. In this method, the change in electrical properties of IC due to laser projection are documented in [10].

Scanning electron Microscope (SEM) is a nondestructive method which can precisely provide images of high resolution and long depth of field. This method is used in military and high reliability areas, but requires a large equipment with Cathode Ray Tube (CRT) and vacuum enclosure.

Charge Induced Voltage Alteration (CIVA) and Light Induced Voltage alteration (LIVA) are imaging techniques used to localize defects. These techniques can either be through electron interaction or photon interaction. LIVA technique detects junction related defects whereas CIVA is used to localize open interconnections [11].
Picosecond imaging circuit analysis (PICA) [12] method was invented at IBM to analyze the emitted photons during device switching. When light source is projected to a Device under Test (DUT), infrared detector captures switching time and the location such that one can localize the Trojan using this fault detection technique.

These physical techniques need a sophisticated equipment and environment which comes at higher cost. For a SoC design with more than 2 billion gates it may take longer time to narrow down the Trojan circuit.

**On chip monitors:**

Another method of Trojan detection after fabrication is on chip monitors. In this method Trojans can be detected either by placing sensors like voltage, current on chip at different locations to monitor the variations or using side channel analysis where Trojans can be detected by analyzing power, current or delay.

**Pre fabrication detection technique**

Detecting Trojan in front end design is a lot easier as adversary has access to netlist level designs.

Detection methods in prefabrication can be divided in two.

1. Fingerprint method
2. ATPG testing method

Fingerprint method is mostly used while detecting Trojans in soft IP cores rather than entire system. In this method, most of designers assume of having a golden IP which is Trojan free design and compares the area, delay and power of golden IP to Trojan circuit. These golden values are known as fingerprints.
ATPG testing method:

Trojans are more prone to the signal that have low controllability and low observability. A Trojan will be a small gate which is used to extract secret key or alters functionality of the output. Using Automatic Test Pattern Generation (ATPG) tools, one can generate patterns for test (PFT) to detect stuck at 0 and stuck at 1 faults. Built in Self-Test (BIST), Design for Test (DFT) are methods to test the circuit with minimal overhead to design. Patterns generated for Trojan free circuit might not satisfy for malicious design as most of Trojans are activated only with special pattern. Trojan detection by DFT and PFT methods give poor results.

1.3. Motivation

With the ever increasing market of electronics, designers are relying on third party IP cores to speed up design time cycle. These third party IP cores can easily insert Trojans. The third party sources share either RTL design or a netlist file of the design. As the number of gates in netlist file is in tens of thousands, even a small change either by inserting or by modifying the netlist can alter the design functionality making the Trojan difficult to detect. Of the available IP cores, major security issues are in cryptographic IP cores. The sole purpose of inserting a Cryptography IP core design is to pass secret information only between sender and a receiver. A simple multiplexed Trojan design can reveal secret data without encrypting it.

None of all the available pre fabrication Trojan detection techniques, detects Trojan at the netlist level. The Trojan insertion at low controllability and observability
signal makes it difficult to detect in ATPG testing. Also fingerprinting method is very difficult as the Trojan area and power are very small.

The motivation of securing third party IP cores leads to implementation of two different cryptographic IP cores named Advanced Encryption Standard (AES) and International Data Encryption Algorithm (IDEA); generating netlist files by using industry standard Cadence RTL compiler tool. AES and IDEA encryption and decryption IP cores functionality are verified on Xilinx Virtex6 FPGA board. A novel library encoding method to detect and nullify the Trojan functionality in the netlist files is implemented in TSMC 0.25um technology. This library encoding method is suitable for all third party IP cores protecting the design even from smaller Trojans.

1.4. Thesis Organization

This thesis is organized as follows: Chapter 2 explains the basics of AES encryption and decryption with an example of data transforming in each and every round. AES encryption and decryption designs are implemented in VHDL and MATAB. Chapter 3 discusses the operation and internal components of IDEA encryption and decryption, these designs are implemented in VHDL and MATAB. Chapter 4 discusses library encoding method and the implementation results. Chapter5 presents a Graphical User Interface designed in MATLAB, embedded AES and IDEA encryption functionalities. The GUI also generates corresponding VHDL files based on user selection. Conclusion and future work are termed in chapter 6.
2. ADVANCED ENCRYPTION STANDARD

With evolution of information, where most applications are on digital data, a secure environment is needed to protect data from eavesdropper. In order to improve confidentiality of data, good cryptographic algorithms are required to encrypt data before transmitting it, in any of possible communication methods. The main purpose of having cryptic techniques is to have secure and authentic communication between both parties. Till date there are various encryption algorithms available in industry, but United States (US) government Federal Information Processing Standards (FIPS) standardizes only algorithms which can be adapted to protect sensitive data.

Of all encryption algorithms, FIPS standardized Data Encryption Standard (DES) as first encryption standard was created in 1977. In Mid-90’s brute force attack on DES made it vulnerable. So in 1997, National Institute of Standards and Technology (NIST) announced a competition to build an Advanced Encryption Standard (AES) as a next encryption standard. After rigorous testing of all submissions, Rijndael was named as an AES standard and named it as FIPS PUB 197 in 2001.

Encryption techniques are broadly divided into symmetric and asymmetric based on key. In symmetric encryption, both parties, sender and the receiver, use same
cryptographic key. On the other hand, asymmetric key also known as public key cryptography have two keys, of which one is private and the other is public.

AES is a symmetric encryption standard by NIST for all sorts of data like email, personal identification numbers and many more [13]. Cryptographic algorithms like AES can be implemented either in software or hardware, depends upon requirements. Software implementations are usually easy to be developed and mostly used in less secure applications, as they are more prone to reverse engineering [14][15]. Another major setback in software implementation is low speed operation. Hardware implementation of cryptographic algorithms is more secure for keys compared to software and also data can operate at higher speeds.

Applications of hardware encryption are on USB memory stick, USB secure dongles and hard drives. Recently, Intel 2010 core processor family has built in hardware cryptographic design which can be accessed through AES new instructions (AES NI)[16].

Three different versions of AES are available based on the key size (128, 192 and 256). This chapter discuss the concepts of AES128 encryption algorithm and hardware implementation in VHDL. Functional verification of AES128 encryption at simulation level in Modelsim and hardware implementation in Xilinx Virtex 6 FPGA board are also included in this chapter.

2.1. AES 128 Overview

Rijndael algorithm was developed by Joan Daemen and Vincent Rijmen to participate in National Institute of Standards and Technology (NIST) competition for data
encryption standard. Of all 15 submissions Rijndael algorithm was named as Federal Information Processing standards (FIPS) publication 197.

In AES encryption, input data are *plaintext* and *key* whereas output is *ciphertext*. AES is a symmetric key algorithm, means same key is used for both encryption and decryption. Symmetric key algorithm is subdivided into stream cipher and block cipher. In stream cipher, each bit is encrypted individually, whereas in block cipher, encryption is done by group of bits. An AES128 is a symmetric block cipher encryption with 8 bit block size. AES128 have 16 blocks where each block is 8 bits.

Three versions of AES are available based on size of the key (128, 192 and 256), plaintext (128 bit). Plaintext is divided into sets of 32 bits called block size Nb = 128/32 = 4. Key is also divided into sets of 32 bits but value varies with version. For AES 192 Nk = 192/32 = 6. Table 2.1 illustrates key length and block size for all versions and relation between Key length Nk and Number of rounds Nr = Nk + 6

<table>
<thead>
<tr>
<th>AES type</th>
<th>Key length Nk</th>
<th>Block size Nb</th>
<th>Number of rounds Nr = Nk + 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES 128</td>
<td>4</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>AES 192</td>
<td>6</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>AES 256</td>
<td>8</td>
<td>4</td>
<td>14</td>
</tr>
</tbody>
</table>

The design algorithm and implementation of this AES128 are discussed in the following sections
For AES128 data input, secure key and data output are 128 bits. As AES is a block cipher, 128 bit data is stored as 4*4 array which is shown in fig 2.1. 128 bit input data is divided into blocks of 8 bits, LSB 8 bits are stored in \( in_0 \) and the next 8 LSB bits in \( in_1 \) and so on.

![4x4 array representation of input 128 bit data](image)

If data input is less than 128 bits, zeros are padded to MSBs. In hardware implementation, as data only deals with binary digits, ASCII data format is used to represent characters. Each entry in 4*4 array is 8 bits, with first 8 LSBs are stored in (0, 0) index and next LSBs are stored in (1, 0) and so on. Once input data is arranged in an array as shown in figure 2.1, 128 bit key is also represented in array by the same manner.

2.2. **Rijndael algorithm basic components**

In order to obtain a cipher text in AES 128, input data and key are passed through initial transformation followed by 10 rounds of transformations. Fig 2.2 explains the sequence of operations in every round. In the initial round plaintext is XORed with Add Round Key and the result is passed to first round. First Nine rounds of AES do four transformation functions: byte substitution, shift rows, mix columns and add round key.
With only three transformations in the final round excluding mix columns step, cipher text is obtained.

Figure 2.2 AES Encryption Algorithm (Source: [17] William stallings pg 154)
2.3. **Rijndael Transformations**
Transformations are the sub components of AES algorithm. The four transformations are

- **Byte Substitution:** A byte will be replaced by the corresponding value of Substitution box array
- **Shift rows:** Shift 1, 2, 3 rows of two dimensional array
- **Mix columns:** Multiply standard array matrix with array output of shift row transformation
- **Key expansion:** Increase the key size by \((N_r +1)*128\) bits or \((N_r+1)*16\) Bytes

2.3.1. **Byte substitution:**
Whenever byte substitution component is called, each and every input of the array is replaced by corresponding value in figure 2.3. Each entry of figure 2.3 is represented in hexadecimal format.
Each and every entry of an array is 8 bits, substitution value is explained with an example.

Input = 10110011

Convert input to Hexadecimal format = B3

Now from B\textsuperscript{th} row and 3\textsuperscript{rd} column in the figure 2.3, the substitution value for B3 is 6D

2.3.2. Shift rows

In this component, rows are circularly left shifted by row number. The starting row number is zero. Figure 2.4 shows row shifting, the zeroth row remains unchanged. Row1 elements are left shifted by 1 element. Similarly row 2 and row 3 are shifted by 2 and 3 elements respectively.
2.3.3. **Mix columns**

This step performs matrix multiplication and logical operations on data array with standard matrix of table 2.2 and data of table 2.3

<table>
<thead>
<tr>
<th>Table 2.2 Standard Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>01</td>
</tr>
<tr>
<td>03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2.3 Data Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>D4 10 60 B0</td>
</tr>
<tr>
<td>CC 21 35 AB</td>
</tr>
<tr>
<td>04 3F 50 90</td>
</tr>
<tr>
<td>32 FF 05 10</td>
</tr>
</tbody>
</table>

Steps to perform Matrix Multiplication are summarized in the following

- Row of one matrix is multiplied with column of the second.
- When multiplying data with 2, do left shift by 1 bit
○ If MSB of data is ‘1’ then after left shift XOR 00011011(1B), substitution box is generated using 1B polynomial.

○ if MSB is Zero then result is the output of left shift

● When multiplying data with 3, do left shift by 1 bit and XOR data

○ conditions for MSB is same as above

● Addition in matrix multiplication can be done with XOR

An example of performing Matrix Multiplication is given below by multiplying first row of standard matrix with first column of Data matrix, result is first element of final matrix

\[
Z(0,0) = 02 \times D4 \ xor \ 03 \times CC \ xor \ 01 \times 04 \ xor \ 01 \times 32
\]

02 \times D4

Convert D4 to binary => 1101 0100, as multiplying by 2 is a left shift
Left shift => 1010 1000
MSB of D4 is 1, XOR the left shift output with “1B”

\[
\begin{align*}
&1010 1000 \\
&0001 1011 \ XOR \\
&1011 0011
\end{align*}
\]

03 \times CC

As 03 = 11 in binary which is same as (10 XOR 01)
\[
(1100 1100 \times (10 XOR 01))
\]

\[
(1100 1100 \times 10) \ XOR (1100 1100 \times 01)
\]

\[
1100 1100 \times 10 = 1001 1000 XOR 0001 1011 (because \ MSB\ of\ CC\ is\ 1)
\]

\[
= 1000 0011
\]

\[
1000 0011 \ XOR 1100 1100
\]

0100 1111

01 \times 04

0000 0100

01 \times 32
0011 0010

\[ Z(0, 0) = 1100 \ 1010 \]

\[ Z(1, 0) = 01*D4 \ xor \ 02*CC \ xor \ 03*04 \ xor \ 01*32 \]
\[ = 1101 \ 0100 \ xor \ 1000 \ 0011 \ xor \ 0000 \ 1100 \ xor \ 0011 \ 0010 \]
\[ = 0110 \ 1001 \]

Final array is represented in table 2.4

<table>
<thead>
<tr>
<th>Table 2.4 final array</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
</tr>
<tr>
<td>69</td>
</tr>
<tr>
<td>46</td>
</tr>
<tr>
<td>CB</td>
</tr>
</tbody>
</table>

2.3.4. Key expansion

As mentioned before AES 128 has 10 rounds of data transformation, the key expansion is needed to assign a new key for every round. Key expansion has to be done for 10 rounds and an initial round with a total of 44 words (176 bytes) and each word width is 32 bits.

![Figure 2.5 Conversion of 8 bit keys to 32 bit words](image)
The first 4 words are derived from initial key. Follow the below sequence of steps to obtain full key expansion

1. For every 4th word i.e. w3, w7 … perform circularly left shift.
   a. Ex: w3 = a2 bc dc 1f => bc dc 1f a2

2. Output of left shift vector is replaced by the byte substitution value
   a. Ex bc dc 1f a2 => 65 86 c0 3a

3. For every round excluding initial round, byte substitution output values are XORed with a RCON vector. Each round has its own RCON vector [17].
   a. Rcon(1) = 01 00 00 00
   b. Rcon(2) = 02 00 00 00
   c. Rcon(3) = 04 00 00 00
   d. Rcon(4) = 08 00 00 00
   e. Rcon(5) = 10 00 00 00
   f. Rcon(6) = 20 00 00 00
   g. Rcon(7) = 40 00 00 00
   h. Rcon(8) = 80 00 00 00
   i. Rcon(9) = 1B 00 00 00
   j. Rcon(10) = 36 00 00 00

   For first round 65 86 c0 3a xor 01 00 00 00 => 64 86 c0 3a = Z1

4. w4, w5, w6 and w7 are obtained as follows [17]
   a. w4 = w0 XOR Z1
   b. w5 = w4 XOR w1
c. \( w_6 = w_5 \text{ XOR} \ w_2 \)
d. \( w_7 = w_6 \text{ XOR} \ w_3 \)

5. To obtain \( w_8 \) value, \( w_7 \) vector is circularly left shifted with its output byte substituted. Output of byte substitution is XORed with Rcon(2). The result is called \( Z_2 \).
   a. \( w_8 = w_4 \text{ xor} \ Z_2 \)
   b. \( w_9 = w_8 \text{ xor} \ w_5 \)
   c. \( w_{10} = w_9 \text{ xor} \ w_6 \)
   d. \( w_{11} = w_{10} \text{ xor} \ w_7 \)

6. Continue the above steps till all \( w_0 - w_{43} \) vectors are derived.

Fig 2.6 explains the detailed components to be used in every round.

![Figure 2.6 Components in a round](image)

**Figure 2.6 Components in a round**

Fig 2.6 with Key expansion component, get the necessary keys to all stages but in an Add round key component of above algorithm, takes 4 words i.e. \( w_0-w_3 \) and convert them back to \( 4*4 \) array as shown in figure 2.7
2.4. AES 128 Design Implementation

AES 128 encryption is implemented in VHDL using top down approach model. Top down encryption is subdivided into small components like byte substitution, shift rows, mix columns and key expansion.

Top level design of AES 128 encryption is shown in figure 2.8.
Byte substitution block level design

Using this component we can design byte substitution for a 4*4 array

Key expansion block design

Figure 2.9 Block level representation of byte substitution

Figure 2.10 Block level representation of key expansion
2.5. Hardware Implementation

2.5.1. FPGA

In the present semiconductor market, the complexity of System on Chip (SoC) increases a lot. To meet existing market demand, Field Programmable Gate Array (FPGA) prototyping is the best way to perform the pre silicon validation and also quickens software development of the system.

Host PC is connected to the FPGA device using JTAG connection port. One can debug the design on FPGA using Chipscope Pro core. Chipscope can access every port through JTAG and cores.

Chipscope basically three types of IP cores.
ICON: Integrated controller core, provides communication between ILA, VIO and JTAG scan port.
ILA: Integrated Logic Analyzer core can be used to monitor any internal signals of the design. Debugger can also set the trigger points.

![Figure 2.11 Design interface with Chipscope cores](image)

Figure 2.11 Design interface with Chipscope cores
VIO: Variable Input and Output core can drive and monitor the signals during run time. The inputs to design are outputs of VIO core whereas the outputs of design are inputs to the VIO core.

The typical block diagram of FPGA debugging and prototyping is shown in fig 2.11. JTAG interface connects the PC and FPGA device. On the FPGA device, design modules can be anything that is to be prototyped. AES, IDEA encryption and decryption IP cores are prototyped on FPGA. ICON and VIO cores are used to feed the data during run time and helps monitoring results.

**AES Encryption IP**

Structural design of AES encryption occupies only 2527 slices and delay of structural design is larger compared to RTL design. The throughput of AES RTL IP core is \((11\times128 \times f \text{ (MHz)})/48\), for clock rate at 294.11MHz (294.11MHz is the maximum clock frequency obtained in Virtex6 FPGA board implementation) throughput is 8627 Mbps. The throughput of structural design is 2167.2 Mbps.

<table>
<thead>
<tr>
<th>Table 2.5 AES encryption Virtex6 board Results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structural Design</strong></td>
</tr>
<tr>
<td>Number of Occupied Slices</td>
</tr>
<tr>
<td>Number of Slice registers</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
</tr>
<tr>
<td>Delay (synthesis report)</td>
</tr>
<tr>
<td>Post PAR Delay</td>
</tr>
</tbody>
</table>
AES Decryption

Structural design of AES decryption occupies only 3642 slices. Delay of structural design is larger compared to RTL design. The throughput of AES RTL IP core is \((11\times 128 \times f (\text{MHz})/58\), for clock rate at 235.84MHz (253.84 MHz is the maximum clock frequency obtained in Virtex6 FPGA board implementation) throughput is 5725.21 Mbps. The throughput of structural design is 1878.04 Mbps.

<table>
<thead>
<tr>
<th></th>
<th>Structural Design</th>
<th>RTL Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Occupied Slices</td>
<td>3642 (9%)</td>
<td>1458 (3%)</td>
</tr>
<tr>
<td>Number of Slice registers</td>
<td>0</td>
<td>1621 (1%)</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>11271 (7%)</td>
<td>4702 (3%)</td>
</tr>
<tr>
<td>Delay (synthesis report)</td>
<td>62.122 ns</td>
<td>Minimum Clock period 4.151ns</td>
</tr>
<tr>
<td>Post PAR Delay</td>
<td>68.156 ns</td>
<td>4.24 ns *</td>
</tr>
</tbody>
</table>

AES Encryption and Decryption FPGA result

Key_in and Data_in are the inputs of AES Encryption cryptographic IP core, Cipher_Data is the output signal.

Cipher_Data and Key_in are the inputs of AES Decryption IP core and Original_Data is the output signal of Decryption core.
2.5.2. ASIC Simulation Results

Simulations results of all the designs are obtained from Mentor Graphics Modelsim 6.6b edition. Script is written in TCL to automate the compiling, simulating and adding waveforms. The example script is shown below

vcom *.vhd

vsim -novopt work.aes_encrypt

add wave -noupdate -radix hexadecimal sim:/aes_encrypt/*

log -r /*
AES Encryption

The simulation waveform of AES encryption on Mentor graphics Modelsim is shown in Fig. 2.13. The waveform shows input key, data, encrypted result and also shows internal round data.

Figure 2.13 simulation result AES encryption

AES Decryption

The simulation waveform of AES decryption on Mentor graphics Modelsim is shown in Fig. 2.14. The waveform shows input key, encrypted data, original result and also shows internal round data.
Figure 2.14 simulation result of AES decryption
3. INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA)

IDEA is the most secured cryptic algorithm and needs less hardware resources compared to AES [18]. The design area of IDEA is smaller when compared to AES. The greatest advantage of IDEA over AES is its 64 bit input data. For small input data in AES, more zeroes are to be padded at data input. This section details about IDEA algorithm and hardware implementation in VHDL. Functional verification of IDEA encryption and decryption at simulation level in Modelsim and at hardware level in Xilinx Virtex 6 FPGA board are also included in this chapter.

3.1. IDEA Algorithm

This section discusses the basic functionality of IDEA encryption and decryption along with its internal components. IDEA mainly has two important characteristics. One is symmetric cryptic process, implying same key is used for both encryption and decryption; another is block cipher where the encryption and decryption data operates in sets of 16bits. In IDEA all input data is represented in 16 bit blocks operating internally.

IDEA cryptography operates on 64 bit input data with a 128 bit key and gives out a 64 bit cipher output. IDEA encryption performs 8 rounds of data manipulations and a final transformation round. Each round uses six 16 bit keys. Input data manipulations are based on those six keys and an additional four block of keys to final round. So, for all rounds, in total 52 keys (8*6 + 4) are needed.
3.2. **IDEA Encryption Components**

Each round has three following basic components:

1. Bitwise XOR: performs bitwise XOR on two 16 bit input blocks.

2. $2^{16}$ modulo addition: Perform binary addition on two unsigned 16 bit data discarding the final carry. The result should not be greater than 65535 in modulo addition.

3. $2^{16}+1$ modulo multiplication: Perform binary multiplication on two unsigned 16 bit data followed by modulo division by 65537, and the final result is 16 bit unsigned data.

![Figure 3.1 key expansion](image-url)
Key expansion: 52 block of keys are generated from a primary input of 128 bit key. First 8 block of keys are generated from 128 bit input key and then circularly left shifted by 25 bits. These 128 bits are used to generate next 8 block of keys. Continue this procedure until 52 block of keys obtained as shown in Fig. 3.1. Table I list keys used in every round of Encryption.

<table>
<thead>
<tr>
<th>Rounds</th>
<th>1st</th>
<th>2nd</th>
<th>3rd</th>
<th>4th</th>
<th>5th</th>
<th>6th</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Key0</td>
<td>Key1</td>
<td>Key2</td>
<td>Key3</td>
<td>Key4</td>
<td>Key5</td>
</tr>
<tr>
<td>2</td>
<td>Key6</td>
<td>Key7</td>
<td>Key8</td>
<td>Key9</td>
<td>Key10</td>
<td>Key11</td>
</tr>
<tr>
<td>3</td>
<td>Key12</td>
<td>Key13</td>
<td>Key14</td>
<td>Key15</td>
<td>Key16</td>
<td>Key17</td>
</tr>
<tr>
<td>4</td>
<td>Key18</td>
<td>Key19</td>
<td>Key20</td>
<td>Key21</td>
<td>Key22</td>
<td>Key23</td>
</tr>
<tr>
<td>5</td>
<td>Key24</td>
<td>Key25</td>
<td>Key26</td>
<td>Key27</td>
<td>Key28</td>
<td>Key29</td>
</tr>
<tr>
<td>6</td>
<td>Key30</td>
<td>Key31</td>
<td>Key32</td>
<td>Key33</td>
<td>Key34</td>
<td>Key35</td>
</tr>
<tr>
<td>7</td>
<td>Key36</td>
<td>Key37</td>
<td>Key38</td>
<td>Key39</td>
<td>Key40</td>
<td>Key41</td>
</tr>
<tr>
<td>8</td>
<td>Key42</td>
<td>Key43</td>
<td>Key44</td>
<td>Key45</td>
<td>Key46</td>
<td>Key47</td>
</tr>
<tr>
<td>Final</td>
<td>Key48</td>
<td>Key49</td>
<td>Key50</td>
<td>Key51</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With the basic components, the following sequence of operations are performed in 1 to 8 rounds of an IP core.

1. Modulo multiplication of Key0 and Data0
2. Modulo addition of Key1 and Data1
3. Modulo addition of Key2 and Data2
4. Modulo multiplication of Key3 and Data3
5. Bitwise XOR of step1 and step3 outputs
6. Bitwise XOR of step2 and step4 outputs
7. Modulo multiplication of step5 output and key4
8. Modulo addition of step 6 and 7 outputs
9. Modulo multiplication of step8 output and key5
10. Modulo addition of step 7 and 9 outputs
11. Bitwise XOR of step 1 and 9 outputs
12. Bitwise XOR of step 2 and 10 outputs
13. Bitwise XOR of step 3 and 9 outputs
14. Bitwise XOR of step 4 and 10 outputs

Figure 3.2 IDEA Internal implementation
The outputs of round 8 is given to concluding round to get the final cipher data as shown in Fig. 3.2. Due to multiple transformation of data in rounds with a 128 bit key, this algorithm is highly secured.

In decryption, encrypted data is recovered by using same key used in encryption step due to symmetric characteristic. In order to recover the cipher data, decryption also uses the same basic components that is used in encryption design but with a different sub key blocks. Sub key blocks to every decryption round are listed in Table 3.2. Comparing Table 3.1 and Table 3.2, to recover the original data, 1\textsuperscript{st} and 4\textsuperscript{th} keys of decryption round 1 are the multiplicative inverse of the final round keys; whereas 2\textsuperscript{nd} and 3\textsuperscript{rd} are additive inverse keys; 5\textsuperscript{th} and 6\textsuperscript{th} keys of round 1 are from round 8 (5\textsuperscript{th} and 6\textsuperscript{th}) keys. Similarly, keys to all rounds can be calculated.

In addition to the above basic components decryption needs additive and multiplicative inverse components to get inversion keys.

Additive inverse of \(2^{16}\) is 2’s complement.

Multiplicative inverse of \(2^{16}+1\) modulo multiplication is complicated and so is calculated using extended Euclidean algorithm [19].
3.3. Design Implementation

IDEA Encryption is implemented using VHDL in top down hierarchal approach. Encryption design is implemented using bitwise XOR, $2^{16}+1$ modulo multiplication and $2^{16}$ modulo addition components. The top level design of IDEA encryption is as shown in fig 3.3

![Figure 3.3 Top Level design of IDEA Encryption](image-url)

Table 3.2 Sub keys to Decryption

<table>
<thead>
<tr>
<th>Rounds</th>
<th>1(^{st})</th>
<th>2(^{nd})</th>
<th>3(^{rd})</th>
<th>4(^{th})</th>
<th>5(^{th})</th>
<th>6(^{th})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Key48(^{-1})</td>
<td>–Key49</td>
<td>–Key50</td>
<td>Key51(^{-1})</td>
<td>Key46</td>
<td>Key47</td>
</tr>
<tr>
<td>2</td>
<td>Key42(^{-1})</td>
<td>–Key44</td>
<td>–Key43</td>
<td>Key45(^{-1})</td>
<td>Key40</td>
<td>Key41</td>
</tr>
<tr>
<td>3</td>
<td>Key36(^{-1})</td>
<td>–Key38</td>
<td>–Key37</td>
<td>Key39(^{-1})</td>
<td>Key34</td>
<td>Key35</td>
</tr>
<tr>
<td>4</td>
<td>Key30(^{-1})</td>
<td>–Key32</td>
<td>–Key31</td>
<td>Key33(^{-1})</td>
<td>Key28</td>
<td>Key29</td>
</tr>
<tr>
<td>5</td>
<td>Key24(^{-1})</td>
<td>–Key26</td>
<td>–Key25</td>
<td>Key27(^{-1})</td>
<td>Key22</td>
<td>Key23</td>
</tr>
<tr>
<td>6</td>
<td>Key18(^{-1})</td>
<td>–Key20</td>
<td>–Key19</td>
<td>Key21(^{-1})</td>
<td>Key16</td>
<td>Key17</td>
</tr>
<tr>
<td>7</td>
<td>Key12(^{-1})</td>
<td>–Key14</td>
<td>–Key13</td>
<td>Key15(^{-1})</td>
<td>Key10</td>
<td>Key11</td>
</tr>
<tr>
<td>8</td>
<td>Key6(^{-1})</td>
<td>–Key8</td>
<td>–Key7</td>
<td>Key9(^{-1})</td>
<td>Key4</td>
<td>Key5</td>
</tr>
<tr>
<td>Final</td>
<td>Key0(^{-1})</td>
<td>–Key1</td>
<td>–Key2</td>
<td>Key3(^{-1})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Key48\(^{-1}\) is multiplicative inverse of Key48
–Key49 is addition inverse of Key49
Key Expansion

Key expansion block takes 128 bit input data and gives 52 sub keys where each sub key is 16 bit wide.

![IDEA Key expansion design block diagram](image)

Figure 3.4 IDEA key expansion design block diagram

Bitwise XOR, modulo addition and modulo multiplication blocks take two 16 bit data and gives out 16 bit data.
IDEA Encryption

The simulation waveform of IDEA encryption on Mentor graphics Modelsim is shown in Fig. 3.5. The waveform shows input key, data, encrypted result and also shows internal round data.

![Simulation waveform of IDEA encryption on Mentor graphics Modelsim](image)

**Figure 3.5 Simulation result of IDEA encryption**

Structural design of IDEA encryption occupies only 1146 slices. Delay of structural design is larger compared to RTL design. The throughput of IDEA RTL IP core is \(\frac{9 \times 64 \times f \text{ (MHz)}}{81}\), for clock rate at 107.52MHz (107.52MHz is the maximum clock frequency obtained in Virtex6 FPGA board implementation) throughput is 764.59 Mbps. The throughput of structural design is 176.24 Mbps.
Table 3.3 IDEA encryption Virtex6 board Results

<table>
<thead>
<tr>
<th></th>
<th>Structural Design</th>
<th>RTL- Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Occupied Slices</td>
<td>1146 (3%)</td>
<td>1196 (3%)</td>
</tr>
<tr>
<td>Number of Slice registers</td>
<td>0</td>
<td>2881 (1%)</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>2867 (1%)</td>
<td>3252 (2%)</td>
</tr>
<tr>
<td>Delay (synthesis report)</td>
<td>250.316ns</td>
<td>Minimum Clock period 9.218ns</td>
</tr>
<tr>
<td>Post PAR Delay</td>
<td>363.148 ns</td>
<td>9.3n</td>
</tr>
</tbody>
</table>

IDEA Decryption

The simulation waveform of IDEA decryption on Mentor graphics Modelsim is shown in Fig. 3.6. The waveform shows input cipher data, key, original data and also shows intermediate data of every round.

![Figure 3.6 Simulation result of IDEA decryption](image-url)
Structural design of IDEA decryption occupies only 1044 slices. Delay of structural design is larger compared to RTL design. The throughput of IDEA RTL IP core is \((9 \times 64 \times f(\text{MHz}))/81\), for clock rate at 108.48MHz (108.48 MHz is the maximum clock frequency obtained in Virtex6 FPGA board implementation) throughput is 771.41 Mbps. The throughput of structural design is 214.1 Mbps.

<table>
<thead>
<tr>
<th></th>
<th>Structural Design</th>
<th>RTL Pipeline design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Occupied Slices</td>
<td>1044 (2%)</td>
<td>956 (2%)</td>
</tr>
<tr>
<td>Number of Slice registers</td>
<td>0</td>
<td>2177 (1%)</td>
</tr>
<tr>
<td>Number of Slice LUTs</td>
<td>2709 (1%)</td>
<td>2912</td>
</tr>
<tr>
<td>Delay (synthesis report)</td>
<td>248.739ns</td>
<td>Minimum Clock period 9.218ns</td>
</tr>
<tr>
<td>Post PAR Delay</td>
<td>298.915 ns</td>
<td>9.3n</td>
</tr>
</tbody>
</table>

Comparing Table 2.4 and 2.5 results with Table 3.3 and 3.4, IDEA consume only one fourth of LUTs but the disadvantage is higher delay. Advantage of IDEA is that, internal computations consume only digital logic whereas AES implementation need more LUTs for s-box implementation.
4. LIBRARY ENCODING

Third party IP core such as cryptography is shared by vendors in netlist format. These netlist files are good place to insert a hardware Trojan. Earlier, section 1 discussed about the types of pre fabrication detection techniques and its disadvantages. This section introduces a new methodology protecting the netlist from inserting a Trojan and also a Trojan detection method at netlist level of chip designing.

4.1. Methodology

The vtvtlib25 standard cell library for TSMC 0.25um technology is used for all netlist generations in ASIC design. This library contains 36 cells of different drive strengths. These 36 cells are divided into groups; based on number of input and output pins.

Following are some of the cells which fall under two input and one output pin group.
and2_1, nand2_1, xor2_1, xnor2_1, mux2_1, or2_1, nor2_1, lp2_1

Library encoding method maps any cell to any other cell, here is a mapping of one such configuration.

<table>
<thead>
<tr>
<th>Cell 1</th>
<th>=&gt;</th>
<th>Cell 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>And2_1</td>
<td>=&gt;</td>
<td>xor2_1</td>
</tr>
<tr>
<td>Nand2_1</td>
<td>=&gt;</td>
<td>xnor2_1</td>
</tr>
<tr>
<td>Xor2_1</td>
<td>=&gt;</td>
<td>and2_1</td>
</tr>
<tr>
<td>Xnor2_1</td>
<td>=&gt;</td>
<td>nand2_1</td>
</tr>
<tr>
<td>Mux2_1</td>
<td>=&gt;</td>
<td>or2_1</td>
</tr>
<tr>
<td>Lp2_1</td>
<td>=&gt;</td>
<td>nor2_1</td>
</tr>
<tr>
<td>Or2_1</td>
<td>=&gt;</td>
<td>mux2_1</td>
</tr>
<tr>
<td>Nor2_1</td>
<td>=&gt;</td>
<td>lp2_1</td>
</tr>
</tbody>
</table>
Whenever the encoding script sees the and2_1 gate, the functionality is modified to xor2_1 gate as per assigned mapping configurations. Fig. 4.1. Shows the block diagram of netlist encoding by library encoding method. The input file to the encoder is a synthesized netlist and the output is a encoded netlist file. Both the input and output files are in verilog hardware description language.

![Netlist Encoding Diagram]

**Figure 4.1 Netlist encoding**

To decode the netlist, the mapped library is remapped to get to the original netlist file as follows.

\[
\begin{align*}
\text{Xor2}_1 & \Rightarrow \text{and2}_1 \\
\text{Xnor2}_1 & \Rightarrow \text{nand2}_1 \\
\text{And2}_1 & \Rightarrow \text{xor2}_1 \\
\text{Nand2}_1 & \Rightarrow \text{xnor2}_1 \\
\text{Or2}_1 & \Rightarrow \text{mux2}_1 \\
\text{Nor2}_1 & \Rightarrow \text{lp2}_1 \\
\text{Mux2}_1 & \Rightarrow \text{or2}_1 \\
\text{Lp2}_1 & \Rightarrow \text{nor2}_1
\end{align*}
\]

Whenever the decoding script sees the xor2_1 gate, the functionality is modified back to and2_1 gate which is the original function of netlist. Fig. 4.2. Shows the block diagram of netlist decoding by library encoding method. Input file to decoder is the encoded netlist and the output is decoded netlist file which is original synthesized netlist file. Both input and output files of netlist decoding are written in verilog hardware description language.
4.2. Trojan Insertion Design

Modifying the netlist of IP by an eavesdropper alters the design functionality. The typical block diagram of the Trojan insertion at netlist level is as shown in Fig 4.3.

The design netlist is encoded using library encoding technique before allowing access to the snooper. Encoded netlist is the only netlist a snooper can access and may insert combinational or sequential type Trojans. Fig. 4.3 shows the Trojan insertion in encoded
netlist file. To recover the design, Trojan inserted in encoded netlist file is decoded to obtain the original netlist. If intruder inserts or modifies the netlist, the Trojan effect is nullified or it may not work in a way the intruder thinks; either can be passing a secret information or activating the trigger. This method of encoding ensures high level of Trojan detection at netlist level for smaller Trojans.

4.3. Trojan Effects in Cryptography IP Cores

Intellectual Property (IP) cores are reusable logic designs that are used to speed up the design cycle of System on chip (SoC). These IP cores are used as building blocks of ASIC designs to meet the market demand. As these IP cores are licensed by third parties, raising the question of security at netlist level.

Of all the available IP cores, security for cryptographic IP cores is of paramount importance. If cryptographic IP cores reveals the secret information then there is no need for having cryptic blocks. Intruders at this level may alter the encryption and decryption designs. So it’s the verification engineer responsibility to verify whether the design is performing what it is supposed to and nothing beyond the design specifications. But the Trojans can be as small as a simple AND gate or as big as a logic counter which can active the trigger to pass on secret the information. Detection of these small Trojans is very difficult due to its smaller size when compared to the design.

Fig. 4.4 shows the block diagram of Trojan insertion in the netlist of AES Encryption IP core. The functionality of AES encryption IP core is to encrypt 128 bit data using 128 secret key and gives output of 128 bit cipher text or data. What if encryption IP core is compromised at netlist level? Fig. 4.4 depicts the Trojan insertion at end of
encryption where the payload is a simple 2*1 Mux and the trigger can be as simple as an AND gate with a select key. The inputs to the payload are cipher text and plain text. Whenever trigger is enabled, Trojan sends the plain text directly without any encryption and if trigger is disabled AES encryption works as a normal encryption IP core without changing the functionality of design.

Fig. 4.5 shows the block diagram of IDEA encryption IP core with the Trojan. IDEA is symmetric type cryptography which encrypts 64 bit plain text using 128 bit secret key and sends out 64 bit cipher text.

Mux Payload has inputs of cipher data and plain data and depend on trigger enabled or disabled Trojan activates sending out the data. The trigger is only activated when the count reaches a specified number in the design and the count is incremented when key meets the specified condition.
The solution to keep the trigger from activation is library encoding. In this method the RTL compiler tool writes encoded netlist file. Intruder may access the encoded file and insert Trojan as shown in Fig.4.5 to bypasses cipher text. The Encoded netlist protects the design from Trojan behavior and alters the Trojan functionality in library decoding.

### Figure 4.5 IDEA Encryption with mux payload

#### 4.4. Synthesis Results

IP cores are synthesized in cadence RTL compiler by running tcl script. Area, power and netlist files are generated.

Synthesis is performed for three different design types to each and every IP cores. The three different design types are Golden IP design, Trojan inserted design and Library encoded designs.

A Golden IP design is the design with no Trojan .i.e. golden IP design gives out the perfect functionality of the IP core.
In Trojan inserted design, the final netlist after synthesis is re-written and inserted. Trojans alter the functionality of the IP core to extract sensitive information.

In Library encoded design method, the final netlist is encoded using encode python script. Script reads the netlist file and encode the gates based on mapping criteria. This method protects the design from hardware Trojans. Decode script re-maps IP core functionality and nullifies the Trojan design.

Area and power results are reported to detect the prefabrication Trojans in fingerprinting method comparing those to detection techniques with library encoding techniques.

4.4.1. IDEA Encryption

Golden IP
Golden IP is the correct functionality of the IP core. Table 4.1 and 4.2 indicates area and power results of golden IP obtained from cadence RTL compiler. Fig. 4.6 shows the result of IDEA encryption on four sets of data.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Cell Area</th>
<th>Net Area</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>idea_encrypt</td>
<td>83725</td>
<td>8073893</td>
<td>3544705</td>
<td>11618598</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Leakage power (nW)</th>
<th>Dynamic power(nW)</th>
<th>Total power(nW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>idea_encrypt</td>
<td>83725</td>
<td>5195.614</td>
<td>29750137675.957</td>
<td>29750142871.570</td>
</tr>
</tbody>
</table>
Figure 4.6 Golden IP IDEA encryption synthesis result

**SET1 input and output data of IDEA Encryption**

**Inputs**
- data\_in: 7FA9\_1C37\_FFB3\_DF05
- Key\_in: 5A14\_FB3E\_021C\_79E0\_6081\_46A0\_117B\_FF03

**Output:**
- Encrypt\_data: 106B\_DBFD\_F323\_0876

**SET2 input and output data of IDEA Encryption**

**Inputs**
- data\_in: 0123\_4567\_89AB\_CDEF
- Key\_in: 5A14\_FB3E\_021C\_79E0\_6081\_46A0\_117B\_FF03

**Output:**
- Encrypt\_data: E8A2\_45E0\_FD18\_FE5C

**SET3 input and output data of IDEA Encryption**

**Inputs**
- data\_in: 7FA9\_1C37\_FFB3\_DF05
- Key\_in: 5214\_FB3E\_021C\_79E0\_6081\_46A0\_117B\_FF03

**Output:**
- Encrypt\_data: 2874\_C20B\_8E29\_96CE

**SET4 input and output data of IDEA Encryption**

**Inputs**
- data\_in: 0123\_4567\_89AB\_CDEF
- Key\_in: 5214\_FB3E\_021C\_79E0\_6081\_46A0\_117B\_FF03

**Output:**
- Encrypt\_data: E0F5\_0F41\_42D8\_8143
Trojan Inserted Design:

A two input Multiplexer is inserted as a Trojan to netlist and the multiplexer payload is activated only with a special key.

### Table 4.3 IDEA encryption Trojan inserted design area result

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Cell Area</th>
<th>Net Area</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>idea_encrypt_trojan</td>
<td>8393</td>
<td>8098371</td>
<td>3557728</td>
<td>11656099</td>
</tr>
</tbody>
</table>

### Table 4.4 IDEA encryption Trojan inserted design power result

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Leakage power (nW)</th>
<th>Dynamic power(nW)</th>
<th>Total power(nW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>idea_encrypt_trojan</td>
<td>8393</td>
<td>5210.85</td>
<td>2992847160.83</td>
<td>9928452371.68</td>
</tr>
</tbody>
</table>

Figure 4.7 Trojan inserted IDEA encryption synthesis result

Area and power results of Trojan inserted design are obtained from cadence RTL compiler tool. Table 4.3 and 4.4 indicates that area and power are increased by 0.003 and 0.006 respectively. Fig. 4.7 shows the result of IDEA encryption on four sets of data. In the Trojan inserted design, output of IP is input data itself for the first two sets as Trojan is activated. IP core works normally, as set3 and set4 keys cannot activate the Trojan.

**SET1 input and output data of IDEA Encryption**

**Inputs**

data_in: 7FA9_1C37_FFB3_DF05  
Key_in: 5A14_FB3E_021C_79E0_6081_46A0_117B_FF03
Output:
Encrypt_data: 7FA9_1C37_FFB3_DF05

SET2 input and output data of IDEA Encryption
Inputs
data_in: 0123_4567_89AB_CDEF
Key_in: 5A14_FB3E_021C_79E0_6081_46A0_117B_FF03
Output:
Encrypt_data: 0123_4567_89AB_CDEF

SET3 input and output data of IDEA Encryption
Inputs
data_in: 7FA9_1C37_FFB3_DF05
Key_in: 5214_FB3E_021C_79E0_6081_46A0_117B_FF03
Output:
Encrypt_data: 2874_C20B_8E29_96CE

SET4 input and output data of IDEA Encryption
Inputs
data_in: 0123_4567_89AB_CDEF
Key_in: 5214_FB3E_021C_79E0_6081_46A0_117B_FF03
Output:
Encrypt_data: E0F5_0F41_42D8_8143

Library Encoding Method
In the library encoding design, if the netlist has any malfunction, the output of IP is completely different (junk) data.

Figure 4.8 library encoding design of IDEA encryption synthesis result
Fig. 4.8 shows the result of IDEA encryption on four sets of data. If the output of library encoding is different from Golden IP method, it indicates there is some Trojan design in the netlist provided by third parties.

**SET1 input and output data of IDEA Encryption**

*Inputs*
- data_in: 7FA9_1C37_FFB3_DF05
- Key_in: 5A14_FB3E_021C_79E0_6081_46A0_117B_FF03

*Output:*
- Encrypt_data: 1029_1835_F323_0804

**SET2 input and output data of IDEA Encryption**

*Inputs*
- data_in: 0123_4567_89AB_CDEF
- Key_in: 5A14_FB3E_021C_79E0_6081_46A0_117B_FF03

*Output:*
- Encrypt_data: 0022_4560_8908_CC4C

**SET3 input and output data of IDEA Encryption**

*Inputs*
- data_in: 7FA9_1C37_FFB3_DF05
- Key_in: 5214_FB3E_021C_79E0_6081_46A0_117B_FF03

*Output:*
- Encrypt_data: 2820_0003_8E21_9604

**SET4 input and output data of IDEA Encryption**

*Inputs*
- data_in: 0123_4567_89AB_CDEF
- Key_in: 5214_FB3E_021C_79E0_6081_46A0_117B_FF03

*Output:*
- Encrypt_data: 0021_0541_0088_8143

**4.4.2. IDEA Decryption**
IDEA decryption recovers encrypted data using a key. Due to symmetry property of IDEA both encryption and decryption use the same key.

**Golden IP**
Golden IP is the correct functionality of IDEA decryption IP core.
Table 4.5 IDEA decryption golden IP area result

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Cell Area</th>
<th>Net Area</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>idea_decrypt</td>
<td>36370</td>
<td>3391005</td>
<td>1497652</td>
<td>4888657</td>
</tr>
</tbody>
</table>

Table 4.6 IDEA decryption golden IP power result

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Leakage power (nW)</th>
<th>Dynamic power (nW)</th>
<th>Total power (nW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>idea_decrypt</td>
<td>36370</td>
<td>2384.673</td>
<td>10816346745.569</td>
<td>10816349130.242</td>
</tr>
</tbody>
</table>

Figure 4.9 Golden IP IDEA decryption synthesis result

Table 4.5 and 4.6 indicates the area and power results of golden IP obtained from cadence RTL compiler. Fig. 4.9 shows the result of IDEA decryption on set1 data.

SET1 input and output data of IDEA decryption

Inputs
- Encrypt_data: 106B_DBFD_F323_0876
- Key_in: 5A14_FB3E_021C_79E0_6081_46A0_117B_FF03

Output:
- Original_data: 7FA9_1C37_FFB3_DF05

The decryption IP core output is primary input data of IDEA encryption.

Trojan Inserted Design:

A two input multiplexer is inserted as Trojan to netlist and the multiplexer payload is activated only with a special key.
Table 4.7 IDEA decryption Trojan inserted design area result

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Cell Area</th>
<th>Net Area</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>idea_decrypt_trojan</td>
<td>36437</td>
<td>3398598</td>
<td>1504511</td>
<td>4903109</td>
</tr>
</tbody>
</table>

Table 4.8 IDEA decryption Trojan inserted design power result

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Leakage power (nW)</th>
<th>Dynamic power(nW)</th>
<th>Total power(nW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>idea_decrypt_trojan</td>
<td>36437</td>
<td>2389.40</td>
<td>1081856640.96</td>
<td>1081859030.37</td>
</tr>
</tbody>
</table>

Figure 4.10 Trojan inserted IDEA decryption synthesis result

Table 4.7 and 4.8 indicates area and power results of Trojan inserted design obtained from cadence RTL compiler tool. Fig. 4.10 shows the result of IDEA decryption on two sets of data. In Trojan inserted design the output of the IP is the first 64 bits of key for the set1 data. In decryption the Trojan is activated based upon input data.

**SET1 input and output data of IDEA decryption**

**Inputs**

Encrypt_data: 106B_DBFD_F323_0876  
Key_in: 5A14_FB3E_021C_79E0_6081_46A0_117B_FF03

**Output:**

Original_data: 5A14_FB3E_021C_79E0

**SET2 input and output data of IDEA decryption**

**Inputs**

Encrypt_data: E8A2_45E0_FD18_FE5C

Original_data: 5A14_FB3E_021C_79E0
Key_in: 5A14_FB3E_021C_79E0_6081_46A0_117B_FF03

Output:
Original_data: 6081_46A0_117B_FF03

Library Encoding Method

In library encoding design, if the netlist has any malfunction, output of the IP is completely different (junk) data.

Fig. 4.11 library encoding design of IDEA decryption synthesis result

Fig. 4.11 shows the result of IDEA decryption on two sets of data. Synthesis result shows the Trojan is present in IP, as the output data of IP core is different from golden IP data.

SET1 input and output data of IDEA decryption

Inputs
Encrypt_data: 106B_DBFD_F323_0876
Key_in: 5A14_FB3E_021C_79E0_6081_46A0_117B_FF03
Output:
Original_data: 4000_4220_0018_7900

SET2 input and output data of IDEA decryption

Inputs
Encrypt_data: E8A2_45E0_FD18_FE5C
Key_in: 5A14_FB3E_021C_79E0_6081_46A0_117B_FF03

Output:
Original_data: 4000_4220_0018_7900
Output:
Original_data: 4000_4220_0018_7900

4.4.3. AES encryption

AES encryption encrypts 128 bit input data with 128 bit key and gives out 128 bit cipher data.

Golden IP

Golden IP of AES encryption is the correct functionality of IP core.

Table 4.9 AES encryption golden IP area result

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Cell Area</th>
<th>Net Area</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_encrypt</td>
<td>227765</td>
<td>1860626</td>
<td>933945</td>
<td>27946421</td>
</tr>
</tbody>
</table>

Table 4.10 AES encryption golden IP power result

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Leakage power (nW)</th>
<th>Dynamic power (nW)</th>
<th>Total power (nW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_encrypt</td>
<td>227765</td>
<td>13938.709</td>
<td>32983951545.678</td>
<td>32983965484.387</td>
</tr>
</tbody>
</table>

Figure 4.12 Golden IP AES encryption synthesis result

Table 4.9 and 4.10 indicate area and power results of golden IP obtained from cadence RTL compiler tool. Fig. 4.12 shows the result of AES encryption on two sets of data. Set data are all represented in hexadecimal format.
SET1 input and output data of AES encryption

Inputs
Data_in: 0123_4567_89AB_CDEF_FEDC_BA98_7654_3210
Key_in: 8976_F7FA_6DDA_7BC0_958E_9D74_9C17_51F0

Output
Cipher_data: 9BF8_4146_34BA_4396_C7FB_3580_A448_B0FF

SET2 input and output data of AES encryption

Inputs
Data_in: 4499_AABB_CCDD_EEFF_FEDC_BA98_7654_3210
Key_in: 9CCB_C949_EBF5_556D_96B8_CA24_194A_FC10

Outputs
Cipher_data: 10EB_866C_4485_E8A3_8A41_A7F8_9BF8_B9F9

Trojan Inserted Design

A two input multiplexer is inserted as a Trojan to bypass the functionality of encryption IP core and the Trojan is activated only with special key.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Cell Area</th>
<th>Net Area</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_encrypt_trojan</td>
<td>228477</td>
<td>18732370</td>
<td>9370214</td>
<td>28102584</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Leakage power (nW)</th>
<th>Dynamic power(nW)</th>
<th>Total power(nW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_encrypt_trojan</td>
<td>22847</td>
<td>14064.52</td>
<td>33676038005.28</td>
<td>33676052069.81</td>
</tr>
</tbody>
</table>

56
Table 4.11 and 4.12 indicate area and power results of Trojan inserted design are obtained from cadence RTL compiler tool. Fig. 4.13 shows the result of AES encryption on two sets of data. In Trojan inserted design the output of the IP is the input for the set1 data.

**SET1 input and output data of AES encryption**

Inputs
Data_in: 0123_4567_89AB_CDEF_FEDC_BA98_7654_3210  
Key_in: 8976_F7FA_6DDA_7BC0_958E_9D74_9C17_51F0

Outputs
Cipher_data: 0123_4567_89AB_CDEF_FEDC_BA98_7654_3210

**SET2 input and output data of AES encryption**

Inputs
Data_in: 4499_AABB_CCDD_EEFF_FEDC_BA98_7654_3210  
Key_in: 9CCB_C949_EBF5_556D_96B8_CA24_194A_FC10

Outputs
Cipher_data: 10EB_866C_4485_E8A3_8A41_A7F8_9BF8_B9F9

**Library Encoding Method**

In library encoding design, if the netlist has any malfunction, output of the IP is completely different (junk) data.
Fig. 4.14 shows the result of AES encryption on two sets of data. In library encoding design if the netlist has any malfunctioned, output of the IP is completely different data which means there is a malfunction design in the netlist. As set1 and set2 output data are not matched with golden IP, the Trojan is present in the netlist.

**SET1 input and output data of AES encryption**

**Inputs**

- Data\_in: 0123_4567_89AB_CDEF_FEDC_BA98_7654_3210
- Key\_in: 8976_F7FA_6DDA_7BC0_958E_9D74_9C17_51F0

**Outputs**

- Cipher\_data: F120_4146_00AA_4186_C6D8_3080_2440_3010

**SET2 input and output data of AES encryption**

**Inputs**

- Data\_in: 4499_AABB_CCDD_EEFF_FEDC_BA98_7654_3210
- Key\_in: 9CCB_C949_EBF5_556D_96B8_CA24_194A_FC10

**Outputs**

- Cipher\_data: F089_8228_4485_E8A3_8A40_A298_1250_3010

### 4.4.4. AES Decryption

AES decryption IP recover the original data from encrypted data using the key. Both encryption and decryption uses same key.
Golden IP

Golden IP is the correct functionality of the IP core.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Cell Area</th>
<th>Net Area</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_decrypt</td>
<td>254187</td>
<td>22952380</td>
<td>10670251</td>
<td>33622631</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Leakage power (nW)</th>
<th>Dynamic power(nW)</th>
<th>Total power(nW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_decrypt</td>
<td>254187</td>
<td>18510.178</td>
<td>48205827628.577</td>
<td>48205846138.755</td>
</tr>
</tbody>
</table>

Table 4.13 AES decryption golden IP area result

Table 4.14 AES decryption golden IP power result

Figure 4.15 Golden IP AES decryption synthesis result

Table 4.13 and 4.14 indicates area and power results of golden IP are obtained from cadence RTL compiler. Fig. 4.15 shows the result of AES encryption on two sets of data. Decryption IP recovered the set1 and set2 cipher data using the encryption “key_in”.

**SET1 input and output data of AES decryption**

**Inputs**

Cipher\textunderscore data: 9BF8\textunderscore 4146\textunderscore 34BA\textunderscore 4396\textunderscore C7FB\textunderscore 3580\textunderscore A448\textunderscore B0FF

Key\textunderscore in: 8976\textunderscore F7FA\textunderscore 6DDA\textunderscore 7BC0\textunderscore 958E\textunderscore 9D74\textunderscore 9C17\textunderscore 51F0

**Output**

Original\textunderscore data: 0123\textunderscore 4567\textunderscore 89AB\textunderscore CDEF\textunderscore FEDC\textunderscore BA98\textunderscore 7654\textunderscore 3210

**SET2 input and output data of AES decryption**

**Inputs**
Cipher_data: 10EB_866C_4485_E8A3_8A41_A7F8_9BF8_B9F9
Key_in: 9CCB_C949_EBF5_556D_96B8_CA24_194A_FC10
Output
Original_data: 4499_AABB_CCDD_EEFF_FEDC_BA98_7654_3210

Trojan Inserted Design

A multiplexer Trojan is added to extract only the key information by disabling the original data output. Trojan is only activated by the special key.

Table 4.15 AES decryption Trojan inserted design area result

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Cell Area</th>
<th>Net Area</th>
<th>Total Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_decrypt_trojan</td>
<td>254318</td>
<td>22967452</td>
<td>10683714</td>
<td>33651166</td>
</tr>
</tbody>
</table>

Table 4.16 AES decryption Trojan inserted design power result

<table>
<thead>
<tr>
<th>Instance</th>
<th>Cells</th>
<th>Leakage power (nW)</th>
<th>Dynamic power(nW)</th>
<th>Total power(nW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>aes_decrypt_trojan</td>
<td>254318</td>
<td>18519.43</td>
<td>48209619496.44</td>
<td>48209638015.88</td>
</tr>
</tbody>
</table>

Table 4.15 and 4.16 indicate area and power results of Trojan inserted design are obtained from cadence RTL compiler. Fig. 4.16 shows the result of AES decryption on
two sets of data. In the Trojan inserted design the output of the IP is the key for the set1 data.

**SET1 input and output data of AES decryption**

**Inputs**

Cipher_data: 9BF8_4146_34BA_4396_C7FB_3580_A448_B0FF  
Key_in : 8976_F7FA_6DDA_7BC0_958E_9D74_9C17_51F0

**Output**

Original_data: 8976_F7FA_6DDA_7BC0_958E_9D74_9C17_51F0

**SET2 input and output data of AES decryption**

**Inputs**

Cipher_data: 10EB_866C_4485_E8A3_8A41_A7F8_9BF8_B9F9  
Key_in : 9CCB_C949_EBF5_556D_96B8_CA24_194A_FC10

**Output**

Original_data: 4499_AABB_CCDD_EEFF_FEDC_BA98_7654_3210

**Library Encoding Method**

In the library encoding design, if the netlist has any malfunction, the output of the IP is completely different (junk) data.

Fig. 4.17 shows the result of AES encryption on two sets of data. In the library encoding design if the netlist has any malfunction the output of the IP changes and the output is
completely different data which mean there is a malfunction design in the netlist. As set1 and set2 output data are not matched with golden IP, the Trojan is present in the netlist.

**SET1 input and output data of AES decryption**

**Inputs**
- Cipher data: 9BF8_4146_34BA_4396_C7FB_3580_A448_B0FF
- Key in: 8976_F7FA_6DDA_7BC0_958E_9D74_9C17_51F0

**Output**
- Original data: 8122_4562_098A_49C0_948C_9810_1414_1010

**SET2 input and output data of AES decryption**

**Inputs**
- Cipher data: 10EB_866C_4485_E8A3_8A41_A7F8_9BF8_B9F9
- Key in: 9CCB_C949_EBF5_556D_96B8_CA24_194A_FC10

**Output**
- Original data: D489_8809_C8D5_446D_9698_8A00_1040_3010

**4.5. Results summary**

Table 4.17 summarizes the fingerprinting and library encoding results of cryptographic IP cores. Area and power fingerprinting method can only specify the additional amount of area and power in IP due to Trojan insertion. The fingerprinting method cannot disable Trojan from activation.

Golden IP column has the results of correct functionality of IP core. Trojan insert column gives the output of Trojan activated design. Library encoding column give the results obtained from library encoding method.
In library encoding method, Trojan area of even a thousandth part of design can be detected. The encrypted or original data output of cryptographic IP core is compared with the Golden IP values. If the output value is different, Trojan is present in the netlist file.

In a Trojan free design the final output is same as golden IP values.

<table>
<thead>
<tr>
<th>IP core</th>
<th>Trojan Area (%)</th>
<th>Trojan Power (%)</th>
<th>Golden IP</th>
<th>Trojan insert</th>
<th>Library Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDEA</td>
<td>0.003</td>
<td>0.006</td>
<td>data_in:</td>
<td>0123_4567_89AB_</td>
<td>data_in: 0123_4567_89AB_</td>
</tr>
<tr>
<td>Encryotion</td>
<td></td>
<td></td>
<td>CDEF</td>
<td>CDEF</td>
<td>CDEF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Key_in:</td>
<td>5A14_FB3E_021C_</td>
<td>5A14_FB3E_021C_</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>79E0_6081_46A0_1</td>
<td>79E0_6081_46A0_1</td>
<td>79E0_6081_46A0_1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Encrypt_data:</td>
<td>E8A2_45E0_FD18_</td>
<td>Encrypt_data: 0123_4567_89AB_</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FE5C</td>
<td>CDEF</td>
<td>CDEF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>0.005</td>
<td>0.02</td>
<td>Data_in:</td>
<td>0123_4567_89AB_</td>
<td>Data_in: 0123_4567_89AB_</td>
</tr>
<tr>
<td>Encryotion</td>
<td></td>
<td></td>
<td>CDEF</td>
<td>CDEF</td>
<td>CDEF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>FEDC_BA9</td>
<td>FEDC_BA9</td>
<td>FEDC_BA9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8_7654_3210</td>
<td>8_7654_3210</td>
<td>8_7654_3210</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Key_in:</td>
<td>8976_F7FA_6DDA_</td>
<td>8976_F7FA_6DDA_</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7BC0_958E_9D74_</td>
<td>7BC0_958E_9D74_</td>
<td>7BC0_958E_9D74_</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>9C17_51F0</td>
<td>9C17_51F0</td>
<td>9C17_51F0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cipher_data:</td>
<td>9BF8_4146_34BA_</td>
<td>Cipher_data: 0123_4567_89AB_</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4396_C7FB_3580_</td>
<td>CDEF_FEDC_BA9_</td>
<td>CDEF_FEDC_BA9_</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A448_B0FF</td>
<td>8_7654_3210</td>
<td>8_7654_3210</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES</td>
<td>8.4e-4</td>
<td>7.8e-6</td>
<td>Cipher_data:</td>
<td>Cipher_data:</td>
<td>Cipher_data:</td>
</tr>
<tr>
<td>Encryotion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In Trojan insertion method, the final netlist can be accessed by intruder and may insert a Trojan design in order to obtain secret information. Trojans bypass IP core functions and output can be either input data or key. Results of Trojan insert column is either a input data or a key.

In Library encoding design, intruder may access encoded netlist and insert a Trojan. To obtain the original function IP core, the encoded netlist is decoded. The Trojan design in an encoded netlist altered by a decode script. Due to functional transformation of Trojan the decoded netlist results are always different from the golden IP values. The library encoding method detects a Trojan and protected IP from transmitting sensitive data like key and input data.
5. GUI APPLICATION FOR CRYPTOGRAPHIC IP CORES

A Graphical User Interface (GUI) is an interface with simple push or a select icon to interact directly with background user functionality. To simplify research work demonstration with a click control application, design a Graphical User Interface (GUI) application in MATLAB. This section details user selection options in GUI and its background functionality.

GUI pane is divided into five sections as shown in Fig.5.1.

1. Cryptography IP core: User can select any one of the four available IP cores

2. Trojan Insertion: User can choose the hardware design either with Trojan or without Trojan.

3. Inputs: User can pass data and key inputs to the IP core in hexadecimal format.

4. Output: The output of corresponding IP core is displayed in hexadecimal format

5. File Generate: User can select either VHDL or MATLAB files to generate

6. Compute: Feed entered inputs to corresponding IP core and give the result in output box
5.1. Cryptography IP core

In this section, users can choose AES encryption, AES decryption, IDEA encryption or IDEA decryption. User can only choose one of the four possible IP cores. Based on user selection the GUI application runs the corresponding IP core functionality in the background.
5.2. Trojan Insertion

Trojan insertion section in GUI determines whether to include Trojan in the hardware designs or not. The Trojan section is useful only in generation of hardware design files. When the user selects “without Trojan” option the design files contain only the IP core functionality. If user selects “with Trojan” then the design files include functionality in addition to IP core. The extra added design can be any malicious design.

5.3. Inputs

Input section of GUI contains two entries. Users can either enter “plain data” or “cipher data” in the First Field and “key” in the second field. For all IP cores the key field width is 32 hexadecimal digits or 128 binary bits. The only acceptable input data format for data and key fields is hexadecimal. In the data field, input string or data length can be either 16 or 32 hexadecimal values. Length of data field depends on the user selection of cryptography IP core, whenever user selects either AES encryption or decryption, the length of data field is 32 and is 16 for IDEA encryption and decryption selection.

5.4. Output

Output section has only one field which gives output of either encryption or decryption IP core. The output for an encryption is called as ciphered data and the output of decryption data is original data or recovered data. The length of output field varies with IP core selection. For AES cores, the length is 32 hexadecimal values, whereas length of IDEA is 16 hexadecimal values.
5.5. **File Generate**

This section generates either VHDL or MATLAB files. Prior to pushing VHDL button, user has to select one of the cryptography IP core, Trojan selection and choose either VHDL or MATLAB push button. If user clicks VHDL push button, in current directory all the VHDL files are generated in the folder named vhdl.

5.6. **Compute**

Compute push button feeds all the data entered in GUI to the background MATLAB script. Script processes input data with the desired IP core functionality and returns the final output of cryptography IP core in the output field.
6. CONCLUSION AND FUTURE WORK

6.1. Conclusion

The proposed pre-fabrication Trojan detection method can detect any small Trojan present in the IP core by library encoding method. Trojan can be detected within a fraction of time than the time taken for ATPG testing.

AES, IDEA encryption and decryption designs are implemented on Xilinx Virtex 6 FPGA evaluation board to ensure hardware compliance. On board debugging is performed using Chipscope IP cores. On virtex6 FPGA hardware, the throughputs of AES encryption, decryption are 8627 Mbps, 5725.2 Mbps respectively and the throughputs of IDEA encryption, decryption IP cores are 764.5Mbps, 771.41Mbps respectively.

The proposed library encoding and decoding scripts are implemented in python. Encoding script accepts netlist file as input maps the existing functionality. Decoding script reads the encoded netlist and remaps the netlist to original functionality. This method detects a small fraction (0.0008% Trojan area) of a design.

Designed a GUI in MATLAB demonstrating all the IP core functionalities and also generating MATLAB and VHDL files of an IP core. All the background scripts are written in python, an executables of python scripts are generated.
6.2. Future Work

The proposed thesis is the Trojan detection at netlist level of design hierarchy in third party IP cores. In this research only combinational Trojans effects with multiplexer payload are considered. As a future work, test the library encoding method on all the kinds of Trojans and verifying the robustness of it can be done. Another future work can be Trojan insertion on other third party IP cores and detecting the Trojan by library encoding. New encoding methods can be introduced to the lower levels of IC designing.
# APPENDIX

AES Encryption

Input data - 4499AABBCDDEEFFEDCBA9876543210

Key input - 9CCBC949EBF5556D96B8CA24194AFC10

<table>
<thead>
<tr>
<th>Round Number</th>
<th>Data Start of Round</th>
<th>After byte substitution</th>
<th>After Shift rows</th>
<th>After Mix Columns</th>
<th>Round Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Round</td>
<td>01 89 FF bb 23 ab ee aa 45 cd dd 99 67 cf cc 44</td>
<td>63 1F A5 15 cc c5 ea 05 F8 5a 13 3F 42 44 40 5d</td>
<td>63 1F A5 15 c5 ea 05 ce 13 3F F8 5a 5d 42 44 40 40</td>
<td>dc 66 e2 79 9a d3 f8 3c 67 4d 87 af c9 70 81 2b</td>
<td>01 42 d6 94 cF ac 55 9c a4 8b 5F bc 91 69 be c9</td>
</tr>
<tr>
<td>round 0</td>
<td>00 cb 29 2F ec 07 bb 36 e1 46 82 25 f6 86 72 8d</td>
<td>77 2d e2 5c 04 03 62 0d 72 08 e5 ae da ac d9 44</td>
<td>77 2d e2 5c 03 62 0d 04 e5 ae 72 08 44 da ac d9</td>
<td>4a 88 56 65 01 da e2 95 69 7d c4 38 f7 14 61 41</td>
<td>de 9c 4a de aa 06 53 cf 79 f2 ad 11 b3 da 64 ad</td>
</tr>
<tr>
<td>round 1</td>
<td>02 fa a8 a7 30 d5 ab f3 1e bf 2a be 7a aa e5 86</td>
<td>77 2d e2 5c 04 03 62 0d 72 08 e5 ae da ac d9 44</td>
<td>77 2d e2 5c 03 62 0d 04 e5 ae 72 08 44 da ac d9</td>
<td>4a 88 56 65 01 da e2 95 69 7d c4 38 f7 14 61 41</td>
<td>de 9c 4a de aa 06 53 cf 79 f2 ad 11 b3 da 64 ad</td>
</tr>
<tr>
<td>round 2</td>
<td>1c 42 d6 3b 29 f4 9f 27 85 63 77 9a 59 60 71 fc</td>
<td>9c 2e f6 e2 a5 bf db cc 97 fb f5 b8 Cb d0 a3 b0</td>
<td>9c 2e f6 e2 Cc a5 bf db f5 b8 97 fb b0 Cb d0 a3</td>
<td>Bc 5d ff 73 4d 99 07 06 19 da 64 54 8e 9a e1 3e</td>
<td>65 af 2f 71 12 3c 41 f3 96 88 3b 99 F6 82 92 2f</td>
</tr>
<tr>
<td>round 3</td>
<td>d9 f2 d0 02 5f a5 46 f5 8f 52 5f cd 78 18 73 11</td>
<td>35 89 70 77 Cf 06 5a e6 73 00 cf bd Bc ad 8f 82</td>
<td>35 89 70 77 06 5a e6 Cf cf bd 73 00 82 Bc ad 8f</td>
<td>2d e6 0f 2b F1 5d 9f 7d 2b 6d 9c 32 89 04 44 53</td>
<td>60 cf e0 91 Fc c0 81 72 83 0b 30 9a 55 d7 45 6a</td>
</tr>
<tr>
<td>round 4</td>
<td>4d 29 ef ba 0d 9d 1e 0f A8 66 ac 9b Dc d3 01 39</td>
<td>E3 a5 df f4 D7 5e 72 76 C2 33 91 14 86 66 7c 12</td>
<td>E3 a5 df f4 5e 72 76 D7 91 14 C2 33 12 86 66 7c</td>
<td>Bc 55 9b de E5 fb 08 68 B2 6e 9c c1 D5 85 02 1b</td>
<td>30 ff 1f 8e 2f ef 6e 1c 81 8a ba 13 D4 03 46 2c</td>
</tr>
<tr>
<td>round 5</td>
<td>8c aa 84 50 Ca 14 66 74 33 e4 26 d2 01 86 44 37</td>
<td>64 ac 5f 53 74 fa 33 92 C3 69 f7 b5 7c 44 1b 9a</td>
<td>64 ac 5f 53 92 74 fa 33 f7 b5 C3 69 9a 7c 44 1b</td>
<td>B0 df 94 48 13 72 7a 1b De 6a 9c d8 8e 91 38 de</td>
<td>8c 73 6c e2 52 bd d3 cf F0 7a c0 d3 Cd ce 88 a4</td>
</tr>
<tr>
<td>round 6</td>
<td>3c ac f8 aa 41 cf a9 d4 2e 10 5c 0b 43 5f b0 7a</td>
<td>Eb 91 41 ac 83 8a d3 48 31 ca 4a 2b 1a cf e7 da</td>
<td>Eb 91 41 ac 8a d3 48 31 4a 2b 31 ca da 1a cf e7</td>
<td>D8 66 a4 f0 E0 4b 4d 13 80 3a 21 92 49 64 3f 73</td>
<td>46 35 59 bb 34 89 5a 95 B9 e3 03 d0 55 9b 13 b7</td>
</tr>
<tr>
<td>round 7</td>
<td>9e 53 fd 4b D4 c2 17 86 39 f9 22 42 1c ff 2c c4</td>
<td>0b ed 54 b3 48 25 f0 44 12 99 93 2c 9c 16 71 1c</td>
<td>0b ed 54 b3 25 f0 44 48 93 2c 12 99 1c 9c 16 71</td>
<td>F6 7a 60 4d F3 fe fc e2 37 fa 0e 41 93 d3 86 fd</td>
<td>Ec d9 80 3b 44 cd 97 02 10 d3 d0 00 Bf 24 37 80</td>
</tr>
<tr>
<td>round 8</td>
<td>1a a3 e0 76 B7 33 6b e0 27 29 de 41 2c f7 b1 7d</td>
<td>A2 0a e1 38 A9 c3 7f e1 Cc a5 1d 83 71 68 e8 ff</td>
<td>A2 0a e1 38 c3 7f e1 A9 1d 83 Cc a5 ff 71 68 c8</td>
<td>E3 67 45 fd E7 1b 1f 4d 41 fb 3b 83 C6 00 e5 cf</td>
<td>80 59 d9 e2 27 ea 7d 7f Dd 0e de de 5d 79 4e ce</td>
</tr>
<tr>
<td>round 9</td>
<td>63 3e 9c 1f C0 f1 62 32 9e f5 e5 5d 9b 79 8b 01</td>
<td>Fb b2 de c0 Ba a1 aa 23 De e6 d9 4c 14 b6 3d 7c</td>
<td>Fb b2 de c0 a1 aa 23 Ba d9 4c De e6 7c 14 b6 3d</td>
<td>64 3d 4e 06 3a d0 ad d2 56 58 86 58 C5 bc f2 3c</td>
<td></td>
</tr>
<tr>
<td>Final</td>
<td>9f 8f 3a c6 9b 7a 8e 68 8f 14 58 be B9 a8 44 01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cipher data is 128 bits represented in hexadecimal
“10eb866c4485e8a38a41a7f89bf8b9f9”
AES Decryption

Input data - 10eb866c4485e8a38a41a7f89bf8b9f9

Key input - 9CCBC949EBF5556D96B8CA24194AFC10

<table>
<thead>
<tr>
<th>Round Number</th>
<th>Data Start of Round</th>
<th>After byte substitution</th>
<th>After Shift rows</th>
<th>After Mix Columns</th>
<th>Round Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Round</td>
<td>9f 8f 3a c6 9b 7a 8e 68 8f 14 58 be B9 a8 44 01</td>
<td>63 3e 9c 1f f1 62 32 c0 e5 5d 9c f5 01 9b 79 8b</td>
<td>63 3e 9c 1f C0 f1 62 32 9c f5 e5 5d 9b 79 8b 01</td>
<td>A2 0a e1 38 C3 7f e1 a9 1d 83 cc a5 Ff 71 68 c8</td>
<td>64 3d e4 06 3a d0 ad d2 56 58 86 58 C5 bc f2 3c</td>
</tr>
<tr>
<td>round 0</td>
<td>Fb b2 de c0 a1 aa 23 Ba d9 4c De e6 7c 14 b6 3d</td>
<td>1a a3 e0 76 33 6b e0 B7 de 41 27 29 7d 2c f7 b1</td>
<td>1a a3 e0 76 B7 33 6b e0 27 29 de 41 2c f7 b1 7d</td>
<td>0b ed 54 b3 25 f0 44 48 93 2c 12 99 1c 9c 16 71</td>
<td>80 59 d9 e2 27 ea 7d 7f Dd 0e de de 5d 79 4e ce</td>
</tr>
<tr>
<td>round 1</td>
<td>E3 67 45 fd E7 1b 1f 4d 41 fb 3b 83 C6 00 c5 cf</td>
<td>9e 53 fd 4b c2 17 86 D4 22 42 39 f9 C4 1c ff 2c</td>
<td>9e 53 fd 4b D4 c2 17 86 39 f9 22 42 1c ff 2c c4</td>
<td>Eb 91 41 ac 8a d3 48 83 4a 2b 31 ca da 1a cf e7</td>
<td>46 35 59 bb 34 89 5a 95 B9 c3 03 d0 55 9b 13 b7</td>
</tr>
<tr>
<td>round 2</td>
<td>F6 7a 60 4d F3 fe fc e2 37 fa 0c 41 93 d3 86 fd</td>
<td>3c ac f8 aa cf a9 d4 41 5c 0b 2e 10 7a 43 5f b0</td>
<td>3c ac f8 aa 41 cf a9 d4 2e 10 5c 0b 43 5f b0 7a</td>
<td>64 ac 5f 53 92 74 fa 33 f7 b5 C3 69 9a 7c 44 1b</td>
<td>8c 73 6c e2 52 bd d3 cf F0 7a c0 d3 Cd ce 88 a4</td>
</tr>
<tr>
<td>round 3</td>
<td>D8 66 a4 f0 E0 4b 4d 13 80 3a 21 92 49 64 3f 73</td>
<td>8c aa 84 50 14 66 74 Ca 26 d2 33 e4 37 01 86 44</td>
<td>8c aa 84 50 Ca 14 66 74 33 e4 26 d2 01 86 44 37</td>
<td>E3 a5 df f4 5e 72 76 D7 91 14 C2 33 12 86 66 7c</td>
<td>30 ff 1f 8e 2f ef 6e 1c 81 8a ba 13 D4 03 46 2c</td>
</tr>
<tr>
<td>round 4</td>
<td>B0 df 94 48 13 72 7a 1b De 6a 9c d8 8e 91 38 de</td>
<td>4d 29 ef ba 9d 1e 0f 0d ac 9b A8 66 39 Dc d3 01</td>
<td>4d 29 ef ba 0d 9d 1e 0f A8 66 ac 9b Dc d3 01 39</td>
<td>35 89 70 77 06 5a e6 Cf cf bd 73 00 82 Be ad 8f</td>
<td>60 cf e0 91 Fc c0 81 72 83 0b 30 a9 55 d7 45 6a</td>
</tr>
<tr>
<td>round 5</td>
<td>Bc 55 9b de E5 fb 08 68 B2 6e 9c c1 D5 85 02 1b</td>
<td>4d 29 ef ba 9d 1e 0f 0d ac 9b A8 66 39 Dc d3 01</td>
<td>4d 29 ef ba 0d 9d 1e 0f A8 66 ac 9b Dc d3 01 39</td>
<td>35 89 70 77 06 5a e6 Cf cf bd 73 00 82 Be ad 8f</td>
<td>60 cf e0 91 Fc c0 81 72 83 0b 30 a9 55 d7 45 6a</td>
</tr>
<tr>
<td>Round</td>
<td>Initial State</td>
<td>Final State</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>---------------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2d e6 0f 2b F1 5d 9f 7d 2b 6d 9c 32 89 04 44 53</td>
<td>d9 f2 d0 02 a5 46 f5 5f 5f cd 8f 52 11 78 18 73 d9 f2 d0 02 5f a5 46 f5 8f 52 5f cd 78 18 73 11 9c 2c f6 e2 Cc a5 bf db f5 b8 97 fb b0 Cb d0 a3 65 af 2f 71 12 3c 41 f3 96 88 3b 99 F6 82 92 2f</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Bc 5d ff 73 4d 99 07 06 19 da 64 54 18 9a e1 3e</td>
<td>1c 42 d6 3b f4 9f 27 29 77 9a 85 63 Fc 59 60 71 1c 42 d6 3b 29 f4 9f 27 85 63 77 9a 59 60 71 fc 77 2d c2 5c 03 62 0d 04 e5 ae 72 08 44 da ac d9 56 ca 80 5e 28 2e 7d b2 Ec 1e b3 a2 Ae 74 10 bd</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4a 88 56 65 01 da e2 95 69 7d c4 38 F7 14 61 41</td>
<td>02 fa a8 a7 d5 ab f3 30 2a be 1e bf 86 7a aa e5 02 fa a8 a7 30 d5 ab f3 1e bf 2a be 7a aa e5 86 63 1F A5 15 c5 ea 05 ce 13 3F F8 5a 5d 42 44 40 de 9c 4a de aa 06 53 cf 79 f2 ad 11 b3 da 64 ad</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>dc 66 e2 79 9a d3 f8 3c 67 4d 87 af e9 70 81 2b 00 cb 29 2F 07 bb 36 ec 82 25 e1 46 8d f6 86 72 00 cb 29 2F ec 07 bb 36 e1 46 82 25 f6 86 72 8d 01 42 d6 94 cF ac 55 9c a4 8b 5F bc 91 69 be e9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Final: 01 89 FF bb 23 ab ee aa 45 cd dd 99 67 cf cc 44

Cipher data is 128 bits represented in hexadecimal
“4499AABCCDDEEFFEFDCA9876543210”
IDEA Encryption

Key in – 8B429D1BF01179C1E09C014445EFA83B

Data in – 0123456789ABCDEF

Sub keys for Encryption

| Key0 – 8B42 | Key26 – 7D41 |
| Key1 – 9D1B | Key27 – DC5A |
| Key2 – F011 | Key28 – 14E8 |
| Key3 – 79C1 | Key29 – DF80 |
| Key4 – E09C | Key30 – 8BCE |
| Key5 – 0144 | Key31 – 0F04 |
| Key6 – 45EF | Key32 – 5EFA |
| Key7 – A83B | Key33 – 83B8 |
| Key8 – 37E0 | Key34 – B429 |
| Key9 – 22F3 | Key35 – D1BF |
| Key10 – 83C1 | Key36 – 0117 |
| Key11 – 3802 | Key37 – 9C1E |
| Key12 – 888B | Key38 – 09C0 |
| Key13 – DF50 | Key39 – 1444 |
| Key14 – 7716 | Key40 – 7168 |
| Key15 – 853A | Key41 – 53A3 |
| Key16 – E707 | Key42 – 7E02 |
| Key17 – 8270 | Key43 – 2F38 |
| Key18 – 0511 | Key44 – 3C13 |
| Key19 – 17BE | Key45 – 8028 |
| Key20 – A0EE | Key46 – 88BD |
| Key21 – 2D0A | Key47 – F507 |
| Key22 – 746F | Key48 – 46FC |
| Key23 – C045 | Key49 – 045E |
| Key24 – E00A | Key50 – 7078 |
| Key25 – 222F | Key51 – 2700 |
### IDEA Encryption

<table>
<thead>
<tr>
<th>Round</th>
<th>Output Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>E599        D74D        190B        2DB6</td>
</tr>
<tr>
<td>2</td>
<td>13C2        1B36        9CBE        6AB3</td>
</tr>
<tr>
<td>3</td>
<td>B696        1A8F        4B3A        45B4</td>
</tr>
<tr>
<td>4</td>
<td>5D15        BB64        3F30        5DB8</td>
</tr>
<tr>
<td>5</td>
<td>91F7        9CDB        A2E4        BD71</td>
</tr>
<tr>
<td>6</td>
<td>0A66        49B4        368D        0792</td>
</tr>
<tr>
<td>7</td>
<td>D603        C351        B129        3ED4</td>
</tr>
<tr>
<td>8</td>
<td>15BC        4430        E353        A077</td>
</tr>
<tr>
<td>Final</td>
<td>C70A        E7B1        B4A8        088E</td>
</tr>
</tbody>
</table>

Encrypted data: C70AE7B1B4A8088E

Sub keys for decryption

<table>
<thead>
<tr>
<th>Key0⁻¹ – E03E</th>
<th>-Key26 – 82BF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Key1 – 62E5</td>
<td>-Key27⁻¹ – CD35</td>
</tr>
<tr>
<td>-Key2 – 0FEF</td>
<td>Key28 – 14E8</td>
</tr>
<tr>
<td>Key3⁻¹ – 7B7B</td>
<td>Key29 – DF80</td>
</tr>
<tr>
<td>Key4 – E09C</td>
<td>Key30⁻¹ – 4AF1</td>
</tr>
<tr>
<td>Key5 – 0144</td>
<td>-Key31 – F0FC</td>
</tr>
<tr>
<td>Key6⁻¹ – 0BED</td>
<td>-Key32 – A106</td>
</tr>
<tr>
<td>-Key7 – 57C5</td>
<td>Key33⁻¹ – 103E</td>
</tr>
<tr>
<td>-Key8 – C820</td>
<td>Key34 – B429</td>
</tr>
<tr>
<td>Key9⁻¹ – 71A7</td>
<td>Key35 – D1BF</td>
</tr>
<tr>
<td>Key10 – 83C1</td>
<td>Key36⁻¹ – 092D</td>
</tr>
<tr>
<td>Key11 – 3802</td>
<td>-Key37 – 63E2</td>
</tr>
<tr>
<td>Key12⁻¹ – 2C24</td>
<td>-Key38 – F640</td>
</tr>
<tr>
<td>-Key13 – 20B0</td>
<td>Key39⁻¹ – 7532</td>
</tr>
<tr>
<td>-Key14 – 88EA</td>
<td>Key40 – 7168</td>
</tr>
<tr>
<td>Key15⁻¹ – 0062</td>
<td>Key41 – 53A3</td>
</tr>
<tr>
<td>Key16 –E707</td>
<td>Key42 –1ֶ –523E</td>
</tr>
<tr>
<td>Key17 –8270</td>
<td>-Key43 –D0C8</td>
</tr>
<tr>
<td>Key18 –1ֶ –D4C7</td>
<td>-Key44 –C3ED</td>
</tr>
<tr>
<td>-Key19 –E842</td>
<td>Key45 –1ֶ –4DC6</td>
</tr>
<tr>
<td>-Key20 –5F12</td>
<td>Key46 –88BD</td>
</tr>
<tr>
<td>Key21 –1ֶ –1BB0</td>
<td>Key47 –F507</td>
</tr>
<tr>
<td>Key22 –746F</td>
<td>Key48 –1ֶ –BF3E</td>
</tr>
<tr>
<td>Key23 –C045</td>
<td>-Key49 –FBA2</td>
</tr>
<tr>
<td>Key24 –1ֶ –B65C</td>
<td>-Key50 –8F88</td>
</tr>
<tr>
<td>-Key25 –DDD1</td>
<td>Key51 –1ֶ –F96A</td>
</tr>
</tbody>
</table>

**IDEA Decryption**

<table>
<thead>
<tr>
<th>Round</th>
<th>Output Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15BC 4430 E353 A077</td>
</tr>
<tr>
<td>2</td>
<td>D603 C351 B129 3ED4</td>
</tr>
<tr>
<td>3</td>
<td>0A66 49B4 368D 0792</td>
</tr>
<tr>
<td>4</td>
<td>91F7 9CDB A2E4 BD71</td>
</tr>
<tr>
<td>5</td>
<td>5D15 BB64 3F30 5DB8</td>
</tr>
<tr>
<td>6</td>
<td>B696 1A8F 4B3A 45B4</td>
</tr>
<tr>
<td>7</td>
<td>13C2 1B36 9CBE 6AB3</td>
</tr>
<tr>
<td>8</td>
<td>E599 D74D 190B 2DB6</td>
</tr>
<tr>
<td>Final</td>
<td>0123 4567 89AB CDEF</td>
</tr>
</tbody>
</table>

Original Data: 0123456789ABCDEF
REFERENCES


