REPUBLIC OF TURKEY
HARRAN UNIVERSITY
GRADUATE SCHOOL OF
NATURAL AND APPLIED SCIENCES

MASTER OF SCIENCE (MSc) THESIS

COMPARISON OF AES AND DES CRYPTOGRAPHIC ALGORITHMS ON FPGA

Abdulsamad Ibrahim Hussein KURD

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ŞANLIURFA
2020

REPUBLIC OF TURKEY
HARRAN UNIVERSITY
GRADUATE SCHOOL OF
NATURAL AND APPLIED SCIENCES

MASTER OF SCIENCE (MSc) THESIS

COMPARISON OF AES AND DES CRYPTOGRAPHIC ALGORITHMS ON FPGA

Abdulsamad Ibrahim Hussein KURD

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ŞANLIURFA
2020

Under the supervision of Assist.Prof.Dr. Nurettin BEŞLİ, this study prepared by Abdulsamad KURD on the subject of **"Comparison of AES and DES Cryptographic Algorithms on FPGA"** was unanimously accepted as MASTER'S THESIS on 21/09/2020 by the following jury in the Department of Electrical and Electronics Engineering at Harran University Graduate School of Natural And Applied Sciences.

Signature

Advisor : Assist.Prof.Dr.Nurettin BEŞLİ .....

e-imzalıdır

Member : Assoc.Prof.Dr.Gökhan GELEN .....

Member : Assist.Prof.Dr.Mehmet Hadi SUZER .....

I Confirm That This Thesis is done in the Department of Electrical and Electronics Engineering and Organized According to the Regulations of Our Institute.

Assoc.Prof.Dr. İsmail HİLALİ
Director of the Institution

Note: The use of original and other sourced reports, charts, figures and photographs without giving reference in this thesis is subject to the provisions of Law No. 5846 on Intellectual and Artistic Works.

CONTENTS

Page Number

ÖZET

Yüksek Lisans Tezi

AES VE DES KRİPTOGRAFİK ALGORİTMALARININ FPGA ÜZERİNDE KARŞILAŞTIRILMASI

Abdulsamad Ibrahim Hussein KURD

Harran Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Dr.Öğr.Üyesi Nurettin BEŞLİ
Yıl: 2020, Sayfa: 83

Bu çalışmanın konusu, mesajları ve verileri anlaşılmaz ve saldırılara karşı dirençli kılmak için dönüştürme sanatı olan kriptografi yöntemleridir. Bu yöntemler, şifreleme ve şifre çözme süreçlerini içerir. Veri üretiminin artan karmaşıklığı göz önüne alındığında, güvenli iletişim ihtiyacı da oldukça hızlı bir şekilde artmaktadır. Paket işleme ve kriptografi dahil olmak üzere iletişim sistemlerinin verimliliği, yalnızca ağın kapasitesine değil, aynı zamanda şifreleme-şifre çözme seviyesine de bağlıdır. Ağ işlemcileri, Uygulamalara Özgü Programlanabilir İşlemcilerdir(ASIC) ve yeni nesil ağ ekipmanlarının kritik bileşenleri haline gelmiştir. Bu çalışmada, iyi bilinen bazı kriptografi yöntemlerini karşılaştıracak ve bu yöntemlerin kripto işlemci tasarımı için avantaj ve dezavantajlarını analiz edeceğiz, İnternetin hızlı ilerlemesi ile siber koruma veri toplama ve iletimi açısından her geçen gün daha da önem kazanmaktadır. Kriptografik algoritmaların kullanılması, hassas verilerin savunulmasını ve siber güvenliğin korunmasını sağladı. En tanınmış şifreleme algoritmaları DES ve AES'tir. İşlemci kapasitesinin artması nedeniyle DES algoritması, AES ile değiştirildi. Şifreleme işlemi, düz metin verilerini şifreli metin adı verilen karıştırılmış ikili verilere dönüştürür. Orijinal metin, şifreli metinden özel bir anahtarla elde edilebilir. Bu çalışma, yazılımda programlanabilen kriptografik algoritmaları analiz etmeyi amaçlamaktadır. Kripto işlemcileri üretmek için Uygulamaya Özel Entegre Devreler (ASIC'ler) ve Alan Programlanabilir Kapı Dizileri (FPGA'ler) kullanılmıştır. Seçilen algoritmalar hızlı ve kolay programlanabilme özellikleri nedeniyle FPGA üzerinde uygulanacaktır. Yerleşik devrenin işlevselliğini kontrol etmek için zamanlama simülasyonu yapılır. Ayrıca yaygın olarak kullanılan şifreleme algoritmaları, Hız, Bellek kullanımı ve Blok kullanımı gibi parametreler açısından karşılaştırıldı. Algoritmalar gömülü ve taşınabilir uygulamalar için Xilinx Spartan 6 Aracı kullanılarak gerçekleştirildi. Sonuçlar gösterdi ki; AES algoritması şifreleme / şifre çözme işlemi için DES'e göre daha az işlem süresine ihtiyaç duymakta, daha az bellek ve işlem bloğu gerektirmektedir.

ANAHTAR KELİMELER: Gelişmiş Şifreleme Standardı (AES), Kriptografi, Kripto işlemci, Veri Şifreleme Standardı (DES) FPGA ve Xilinx.

ABSTRACT

MSc Thesis

COMPARISON OF AES AND DES CRYPTOGRAPHIC ALGORITHMS ON FPGA

Abdulsamad Ibrahim Hussein KURD

Harran University
Graduate School of Natural and Applied Sciences
Department of Electrical Electronic Engineering

Supervisor: Assist. Prof. Dr. Nurettin BEŞLİ
Year: 2020, Page: 83

The topic of this study is the methods of cryptography, which is the fine art of transforming messages or data in order to render them unrecognizable and resistant to attacks. It involves encryption and decryption processes. Given the growing complexity of data production, the need for safe communication is also rising quite rapidly. The efficiency of communication systems, including packet processing and cryptography, relies not only on the capacity of the network but also on the level of encryption-decryption. Network processors are programmable processors unique to applications and have become critical components of next-generation network equipment. In this study, we will compare some well-known cryptography methods and analyze the advantages and disadvantages of these methods for crypto-processor design. With the fast progression of the internet, in terms of data collection and delivery, cyber protection is becoming extremely relevant every day. The use of cryptographic algorithms has accomplished the protection of sensitive data and the preservation of cyber security. Most recognized cryptographic algorithms are DES and AES. Due to the growth of processor capability, the DES algorithm is replaced with AES. Encryption operation changes a plain text data to scrambled binary data called ciphertext. The original text can be retrieved with an special key from the ciphertext. This study aims to analyze cryptographic algorithms that can be programmed in software. Application Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) have been sed to produce crypto-processors. The chosen algorithms will be implemented on FPGA due to their fast and easy programming capabilities. Timing simulation is conducted to check the built circuit's functionality. Furthermore, the commonly used encryption algorithms are also compared in terms of the parameters such as Speed, Memory usage and Block usage. The algorithms were implemented using the Xilinx Spartan 6 Tool for embedded and portable applications. The results show that AES needs less processing time than DES, requires less memory and blocks for encryption/decryption process.

KEYWORDS: Advance Encryption Standard (AES), Cryptography, Crypto-processor, Data Encryption Standard (DES) FPGA, and Xilinx.

ACKNOWLEDGEMENT

I am grateful to my advisor, Assist.Prof. Dr.Nurettin BEŞLİ, who assisted me in every stage of this thesis from the beginning to the end and meticulously conveyed his knowledge and experience, to my colleagues at the Department of Electrical and Electronics Engineering, especially to Prof.Dr.Ali KIRÇAY, to Electrical and Electronics Engineer Ahmed SHERWANY, and my endless thanks to my family and all my friends, who are always with me with their material and moral support.

LIST of FIGURES

Page Number

LIST of TABLES

Page Number

ABBREVIATIONS

3DES	Triple-DES (Data Encryption Standard)
AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
BRAM	Block RAM
CLB	Configurable Logic Block
CMP	Compare
CMT	Clock Management Tile
CPU	Central Processing Unit
CTI	Cipher text Input
CTO	Cipher text Output
CTR	Counter mode
CTRL	Control
DCM	Digital Clock Manager
DES	Data Encryption Standard
DI	Data Input
DO	Data Output
DPRAM	Dual Port RAM
DRM	Digital Rights Management
EM	Encryption Modes
FF	Flip-Flop
FIFO	First In – First Out memory structure
FIPS	Federal Information Processing Standards
FPGA	Field Programmable Gate Arrays
GF (2^8)	Galois Field for 2^8
HDL	Hardware Description Language
I/F	Interface
I/O	Input/Output
ICAP	Internal Configuration Access Port
Inv S-Box	Inverse Substitution byte operation
Inv Sub Bytes	Inverse Substitute Bytes operation
InvMixColumns	Inverse Mix Columns operation
IP	Intellectual Property
K	Key
KB	KiloBytes
KI	Key Input
KN	Key of round
KO	Key Output
LBA	Logic Block Array
Mb	Megabit
MB	MegaBytes
MC	Mix Columns (in AES)
MCCP	Multi-Core Crypto-Processor
ModelSim	VHDL/Verilog simulator
Nr	Number of round
PC	Personal Computer
PK	Public Key
PLD	Programmable Logic Device
PT	Plain text
PTI	Plain text Input
PTO	Plain text Output
R1	Round number one
RAM	Random Access Memory
RK	Round Key
ROM	Read-Only Memory

S	Seconds
S-Box	A lookup table that holds non-linear substitute byte values
SL	Shift to Left (in DES)
SRAM	Static RAM
USB	Universal Serial Bus
*	Multiplication operation
$\oplus$	Exclusive-OR operation
•	Matrix Multiplication operation

1. INTRODUCTION

Cryptography is a way of protecting knowledge and correspondence through the use of protocols. Information can be understood only by those to whom the data is intended. In addition, cryptography can be applied to communication methods derived from variety of protocols of the networking package. Cryptography is a series of rule-based equations called algorithms, converting plain messages to hard-to-decipher data. These deterministic algorithms are used for key generation, authentication, digital signature, data protection assurance, internet surfing, and sensitive communications (Setyaningsih and Wardoyo, 2017).

In modern times, cryptography focuses on communications security i.e., encryption, transmission and decryption of messages, thus making them unreadable by interceptors or eavesdroppers without confidential details (namely the key required to decrypt the message). Encryption aims at protecting items that should be hidden. The research area has grown in recent decades beyond protection issues to include methods for message validity verification, preservation of the sender/receiver identification, digital signatures, virtual proof and safe computation (Saurabh and Divya, 2017).

In the computer period, Cryptography has been primarily concerned with textual and lexicographic trends up until the early 20th century. Since then, the focus has changed and cryptography makes comprehensive use of mathematics, including facets of mathematical analysis, multiplication of matrixes, probability, combinatory, logical arithmetic, algebraic topology, and typically multivariable calculus.

Additionally, processors facilitated encryption of any kind of data evenly divided in any binary form, whereas conventional ciphers only encrypted written language texts; this was new and significant. Text cryptography has been substituted by the usage of machines for both cipher creation and cryptographic algorithms.

Most computer ciphers can be represented by their operations on logic circuits (sometimes in groups or blocks) as opposed to classical ciphers which typically regulate phonetic symbols (i.e. letters and digits). Computers, therefore, supported cryptographic techniques and often compensated the extra difficulty of the cipher. Strong modern ciphers have stayed ahead of computer capability. It is generally the case that utilizing a standard cipher is quite powerful i.e. encryption is simple and need of little resources, such as memory or CPU capacity, whereas, decryption requires an effort of enormous magnitude, rendering cryptographic algorithms very costly (Jeeva et al., 2012).

1.1. Cryptography Goals

When utilizing cryptography, several objectives such as Confidentiality, Authentication, Integrity, Non-Repudiation and Availability can be achieved. Either all or a few objectives may be realized in one program.

- Confidentiality: Knowledge in the transmitted data is only accessible by authorized parties.
- Authentication: Confirmation of the identification is correctly established.
- Integrity: Distributed or processed knowledge may be changed only by registered parties.
- Non-Repudiation: The communication cannot be refused by either the sender or the recipient of the document.
- Availability: Approved parties may access computer system assets as required.

1.2. Network Processor

The operation of the network is carried out by an automated circuitry and has a function set directly targeting the field of the communication program. Network processors are usually configurable computing machines that should also provide

standard features close to central processing systems and are widely found in various forms of appliances and artifacts. Data (voice, video, data) is transmitted in digital communications systems as stream traffic (static packet switching) as opposed to traditional communications structures transmitting content as analog signals, such as in the public telecommunications network (PSTN) or analog TV/radio networks (Guleria and Vatta, 2013; Buono et al., 2014).

As for the Architectural Model of Network Processors, many architecture paradigms are widely used to work with large data-rates. One of them is the pipelined processor architecture in which each step of the tasks is implemented by a segment with different parts of streaming data. The other one is multiprocessor architecture, often requiring multithreading. Processors can be designed with advanced microcode engines to efficiently perform the task at hand (Yang, 2011).

1.3. Cryptographic Forms

1.3.1. Symmetric Key

Symmetric key cryptography shown in Figure 1.1 uses the same key to encrypt information as to decrypt the data. DES, AES and RC4 can be named as among the Symmetric key algorithms. Usually, this makes it easier than public-key encryption (Kim, 2012). The concern with this encryption approach is that the key must be present in order for information to be decrypted. This triggers two questions: The first issue is that you need to safely store the key (Alioto and Rocchi, 2010). If an intruder were to obtain this key, all data encrypted by the key might be decrypted. It is normal to store Symmetric keys in a secure place and only to use them when necessary. The next question is that the information can be decrypted only by an authorized entity. A protected channel must be used to transfer the key for this to occur. The secret key must be also delivered using standardized acceptable methods (Agrawal and Mishra, 2012; Bhatele, 2012).

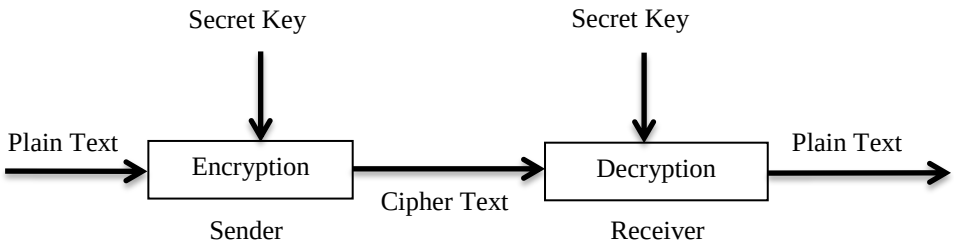

Figure 1.1. Symmetric Key Cryptography

1.3.2. Asymmetric Key

Two keys, public and private keys, are used for public key encryption/decryption. To explain, assume that two persons wish to engage with one another. There is a third person in the middle, attempting to eavesdrop on their conversation. For standard encryption that uses the same key, without the 3rd party having the key, the challenge is getting the key to the second party. The public key is essential for public-key encryption to encrypt traffic, but it does not need to be protected. If the public key was accessed by a 3rd party, they would not be able to access the encrypted data using the public key. You need a private key in order to decrypt the data. The secret key has to be carefully protected. The major benefit is that encryption takes place without the private key. This suggests that it is never necessary to pass the private key and there is also no risk that a 3rd party will access the public key. Asymmetric Key Algorithms can be named as RSA, DHA, DSA, MD5 and ECC (Agrawal and Mishra, 2012; Alammd and Khan, 2013).

Consumers and applications must be sure that indeed a security key is legitimate, belongs to the individual or the organization intended, and has not been compromised or changed by malicious third parties. There is no ideal answer for such a problem of public key cryptography. A mutual key implementation, where trustworthy security authorities check control of key pairs and keys, is the most popular technique, but cryptographic approaches focused on the Very Strong Privacy paradigm, rely on a hierarchical encryption scheme called a trustworthy network,

relying on individual authorships for User-Public Key connections shown in Figure 1.2. (Mahajan and Sachdeva, 2013).

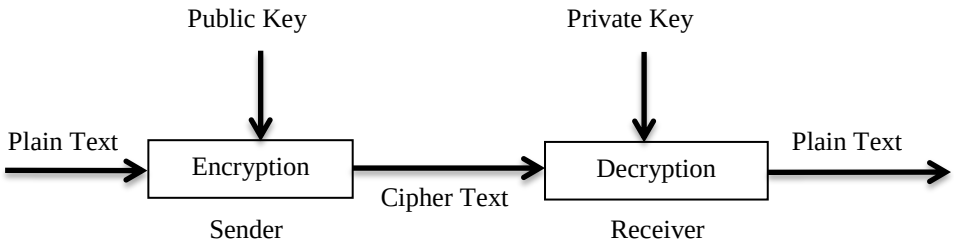

Figure 1.2. Asymmetric Key Cryptography

1.4. Project Plan

Algorithms of AES and DES, the encryption standards accepted by NIST, are now by far the preferred option for encryption in networked applications. Implementing the algorithms on hardware provides higher efficiency but less versatility and is often complicated and time-consuming relative to implementing by software.

Our goal is to implement the Advanced Encryption Standard (AES) and Data Encryption Standard (DES) on hardware using an FPGA chip. While FPGA provides an efficient design, the design can be improved and updated by software. Hardware architecture will be developed by utilizing the Xilinx ISE Design Suite 14.7.

1.5. Project Modeling

This diagram shows the structure of our study on the Advanced Encryption Standard (AES) and Data Encryption Standard (DES) cryptographic algorithms. Project Models have been developed using Xilinx ISE Design Suite 14.7 platform first and implemented on Xilinx Spartan-6 Family XC6SLX45 Field Programmable

Gate Arrays (FPGAs). In addition, the encryption and decryption algorithms are compared shown in Figure 1.3.

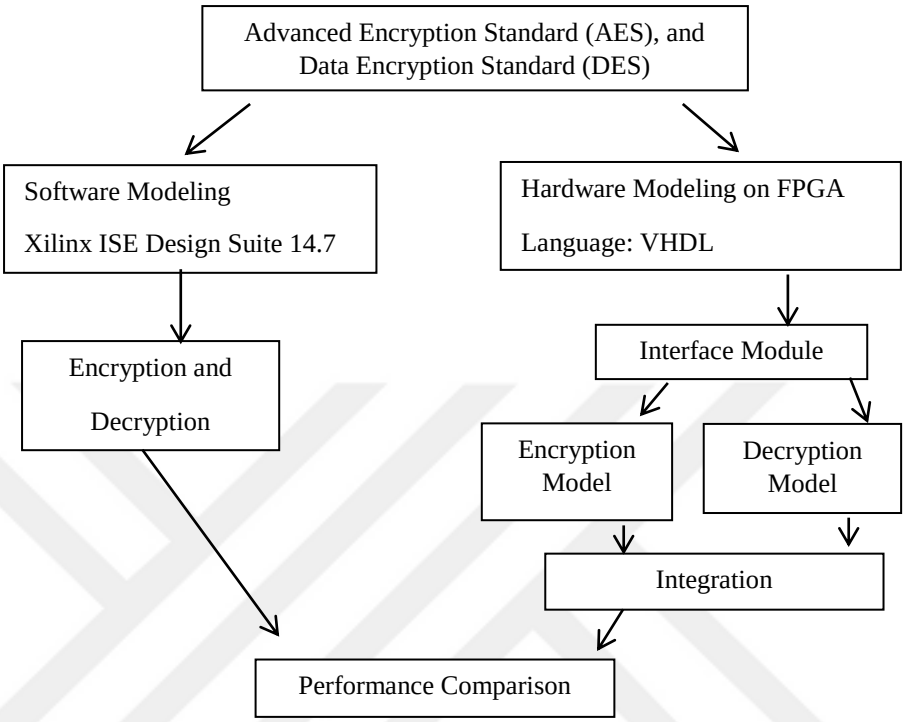

Figure 1.3. Project Modeling

2. LITERATURE REVIEW

Srinivas and Shanbhag (2014) presented a comparative study of DES and BLOWFISH symmetric key cryptography algorithms with differences in parameters such as various types of data, data size and key size. To demonstrate the efficiency of the algorithms by adjusting some of these parameters, the experimental work was conducted on DES and BLOWFISH Algorithms. The execution time was analyzed as a function of the length of the encryption key and the file size.

Singhal and Raina (2011) in their comparative study examined the AES and RC4 for better use of algorithms. Results reveal the contrasting and distinctive estimates and address encoding time and decoding time, memory usage and CPU processor time. They compare the AES algorithm (block cipher) to the RC4 algorithm (stream cipher) in different operation modes and in different settings such as variable key size and different packet data size. Test Results suggest that the RC4 algorithm is faster than AES and RC4 is stronger than AES.

Bhanot and Hans (2015) examined algorithms such as DES, TDES, RSA, AES, BLOWFISH, and RC5. They stated how these algorithms vary in key length, protection levels and resistance to attacks. The results showed that keys with more number of bits need more computing time. BLOWFISH has no hazard, so the RSA and BLOWFISH are stronger.

Ghodke and Mali (2016) in their research, realized the Field Programmable Gate Array(FPGA) based hardware implementation of the AES-Rijndael encoding and decoding process. With respect to FPGA and VHDL, this study explores the AES algorithm. The emulation and modification of the synthesizable VHDL code is accomplished. To reduce resource usage, all transformations of both encryption and decryption were modeled using an incremental design concept. The pipeline

architecture on FPGA for the AES algorithm has been suggested, which provides high resource efficiency.

Pritamkumar and Vrushali (2015) researched the low-area implementation of the AES algorithm on FPGA. It is very easy to process digital information today, but it allows unauthorized users to access this information. The key goal of the paper is to apply the AES algorithm on a reconfigurable framework easily and safely. The AES algorithm is designed to achieve lower power usage and higher throughput consumption.

Riman and Abi-Char (2015) carried out a Comparative Analysis of Block Cipher-Based Encryption Algorithms. In order to secure shared data, several strategies are needed. AES, DES, 3DES and E-DES encryption techniques were studied in this paper. A comparative study was carried out on the symmetric encryption algorithms mention above with regard to processor time, size of the block, size of the key, memory and power usage. The results of the experiments are given to evaluate the efficacy of each algorithm. Educational-DES performed better than DES, 3DES and AES algorithms.

Gurpreet and Supriya (2013) give a survey of common encryption algorithms (RSA, DES, 3DES and AES) for Information Security. The robust Encryption methods are being implemented by multiple organizations in the field of Information Management.

Abdulwahid et al. (2018) presented a Comparison of Cryptographic Algorithms: DES, 3DES, AES, RSA and BLOWFISH for Guessing Attacks Prevention. In this paper, attributes, deficiencies and similarities of cryptographic algorithms have been given. Each cryptographic algorithm has its own strong and weak points.

Brindha et al. (2014) presented a study on use of Symmetric Algorithm for Image Encryption. Encryption is a tool for shielding data from getting damaged by using specific algorithms and keys before delivering it across the network. The decryption keys are used to bring back the initial digital data from the encrypted data that is transmitted. This paper provides an overview of the DES algorithm. The suggested method would replicate the original picture with no loss of details. This paper also carries out a comparative analysis of the DES algorithm with new picture encryption algorithms.

Soni et al. (2012) presented a study on Analysis and Comparison of AES and DES Cryptographic Algorithms. An overview and evaluation of different parameters of DES and AES encryption systems are provided in this article.

3. MATERIAL and METHOD

3.1. AES (Advanced Encryption Standard (Rijndael))

The AES (Advanced Encryption Standard) is developed in 1998 by Vincent Rijmen and Joan Daemen (Abdulwahid et al., 2013), The AES transformation process involves steps such as Key expansion, Add round key, Sub Bytes, Shift Rows and Mix columns in forward and reverse order as seen in Figure 3.1. In the beginning, Message and Key are determined. Then Key expansion process is implemented to produce 128,192 or 256-bit Round key for each round according to the number of rounds (Nr). Key State also called Key 0 is used for Add Round Key operation in the initial round.

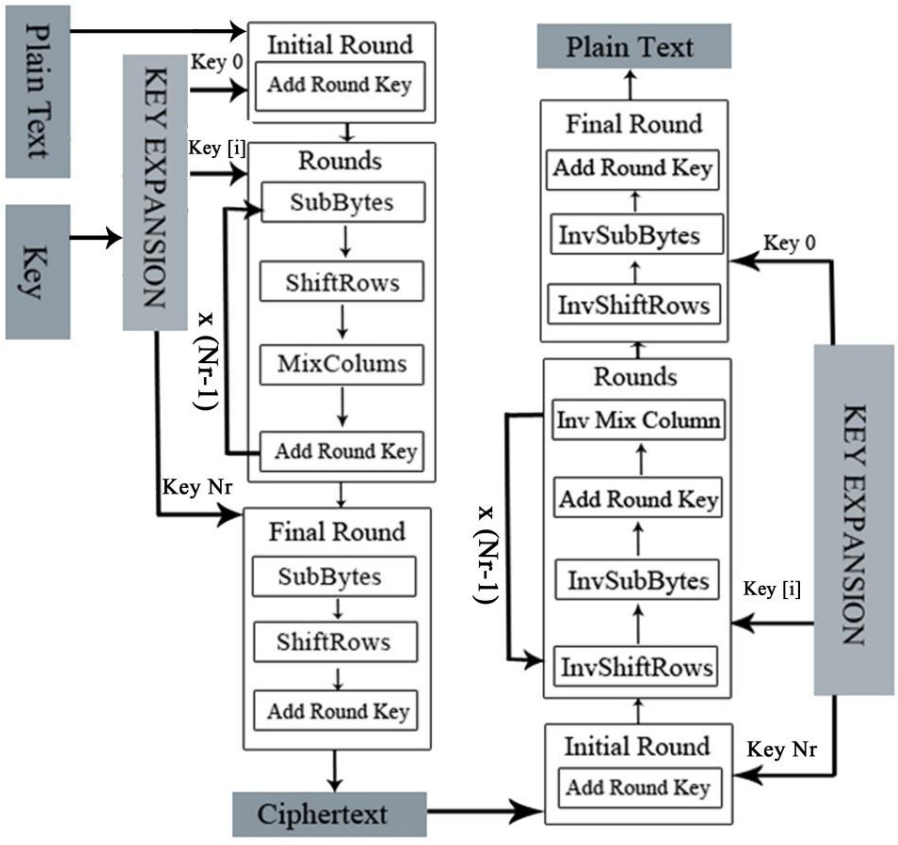

Figure 3.1. Overview of AES

In the Rijndael method, the number of rounds to perform depends on both AES bit sizes and the key length. The main numbers are 128 (10 rounds), 192 (12 rounds), and 256 (14 rounds) (Prashanti et al., 2013).

Table 3.1. The AES Key length

AES (bits)	*Key length (words)*	*Number of rounds(Nr)*
128	4	10
192	6	12
256	8	14

3.1.1. Encryption Block for AES Algorithm

At the encryption block, Key Expansion and the initial round are completed first and rounds according to Table 3.1 are carried out. Each round except Final round consists of four steps: Sub Bytes, shifting row, mix column and add round key. The final round has three steps excluding mix column. The encoding process with a data-size of 128-bit passes through one initial round and 10 rounds. In the initial round, both the input message and the key are used to generate an outcome through only Add Round Key operation. After the Initial Round, it will go through the transformation of the Sub Byte, Shift rows, Mix columns, and finally adding the special round key generated for each round (Jeeva et al., 2012; Sakiyama et al., 2011).

3.1.1.1. Key Generation

In the Key Generation step, each round of Key Expansion involves 4 activities: Rotation, Substitution, Round Constant and XOR operation. Key State, also called Key 0, is formed from the original cipher key. The first 4 bytes of the key become the first column of the key state and the second 4 bytes become the second column and so on. The key state of 4 Rows × 4 Columns × 8 bits = 128 bits is

created. In order to generate 10 sub keys for each round, the key state, sub-byte and Rcon will be used. As an example, the 128-bit original cipher key is given below in hexadecimal format.

128-bit Key: AB12C3D03492E87F5247AD86BA63E81F

Table 3.2. Key (128-bit)

AB	12	C3	D0	34	92	E8	7F	52	47	AD	86	BA	63	E8	1F

The first 4 bytes of the key became the first column of the key state and the second 4 bytes become the second column and so on as shown below in Table 3.3.

Table 3.3. Key state

AB	34	52	BA
12	92	47	63
C3	E8	AD	E8
D0	7F	86	1F

3.1.1.2. Rotating the word

In this step, in order to generate the Rotated Word, the last column of the key state is rotated upward, and the top cell goes to the bottom of the last column as shown below in Table 3.4.

Table 3.4. Rotate word of last Colum

AB	34	52	BA
12	92	47	63
C3	E8	AD	E8
D0	7F	86	1F

AB	34	52	63
12	92	47	E8
C3	E8	AD	1F
D0	7F	86	BA

3.1.1.3. Sub Byte of Rotated word and S-Box

In the substituting byte operation with the S-Box, the first hexadecimal character of each key state became row number and the second one became column number and the intersection point on the S-Box table given in Table 3.5 became new byte (Ahmad et al., 2010).

In other words, the actual byte of states is converted into a specific byte as follows; the left 4 bits of the byte are used as a row value and the right 4 bits are used as a column value. These values serve as indices for the row and column to pick a specified 8-bit output value in the S-Box is shown in Table 3.5 and the new Sub Byte values are shown in Table 3.6.

Table 3.5. S-Box

	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***A***	***B***	***C***	***D***	***E***	***F***
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Table 3.6. Sub Byte

AB	34	52	63	→	62	28	48	FB
12	92	47	E8		39	74	16	9B
C3	E8	AD	1F		33	C8	18	C0
D0	7F	86	BA		FC	6B	DC	F4

3.1.1.4. Round Constant Array

The round constant word array, Rcon, is given in Table 3.7. Rcon[1] contains the values in the first column of the Rcon array. The Round constant values appear as (01, 00, 00, 00) for the first round and (02, 00, 00, 00) for the second Round and so on. The Rcon is a predefined table for key generation in AES as shown below in Table 3.7.

Table 3.7. The Content of the Rcon (Round)

Round	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Rcon	01	02	04	08	10	20	40	80	1B	36
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00
	00	00	00	00	00	00	00	00	00	00

In order to generate 10 new sub keys, The Key State, SubByte and Rcon will be combined with XOR operation as shown in Figure 3.2. The first column of the Key state (Key 0) and the last column of Sub Byte and Rcon[1] will be combined with XOR. The result will be the first column of the Key 1 for Round 1. Then, the second column of Key 0 will be XORed with the first column of Key 1 to produce the second column of Key 1. This operation will be done two more times to form the third and the fourth column of Key 1. After creating Key 1, the same operations will

be repeated using Key 1, Sub Byte and Rcon[2] to generate Key 2 and so on, as shown in Figure 3.2.

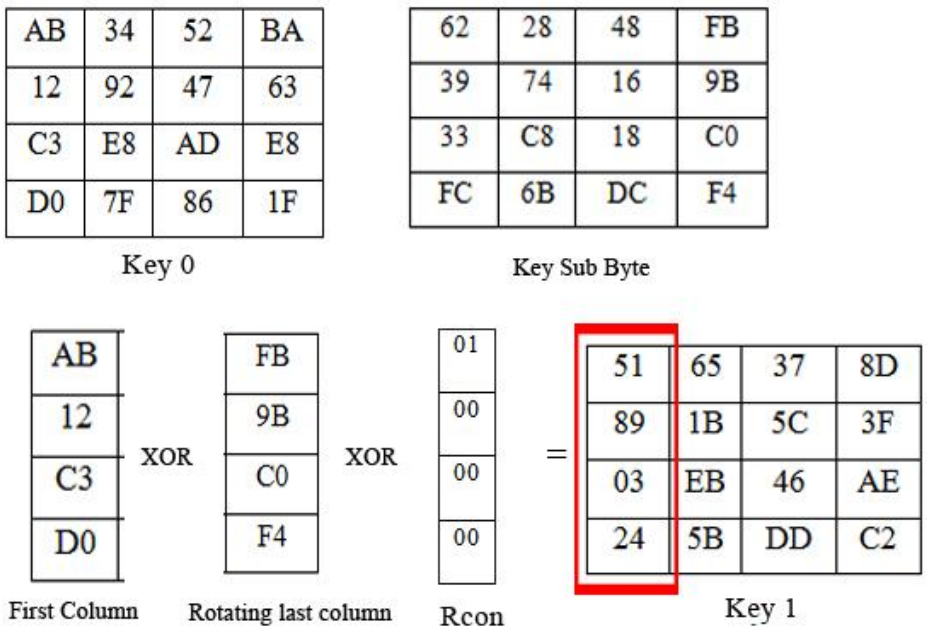

Figure 3.2. Sub Key Generations

Key 1: 51890324651BEB5B375C46DD8D3FAEC2

Key 2: 266D26794376CD22742A8BFFF915253D

Key 3: 7B5201E03824CCC24C0E473DB51B6200

Key 4: DCF86235E4DCAEF7A8D2E9CA1DC98BCA

Key 5: 11C51691F519B8665DCB51AC4002DA66

Key 6: 46922598B38B9DFEEE40CC52AE421634

Key 7: 2AD53D7C995EA082771E6CD0D95C7AE4

Key 8: E00F54497951F4CB0E4F981BD713E2FF

Key 9: 86974247FFC6B68CF1892E97269ACC68

Key 10: 08DC07B0F71AB13C06939FAB200953C3

3.1.1.5. Rounds Operations

The Rijndael cryptography with 128-bit data block passes through one initial round and 10 rounds of encoding. The initial round starts with the formation of the Message state from the plaintext. After that, addition of the Message state and Round key is implemented with XOR operation. The result is applied to the SubByte in the first round. Shiftrow receives the result of the SubByte. Then, the output of Shiftrow is applied to Mixcolumns which implementes multiplication in the Galois Field. Finally, the addition of the MixColumns state and Round key with XOR is carried out. Figure 3.3 shows the round activities (RoundKey, SubByte, ShiftRows and MixColumns) in the AES encryption algorithm. In the final round, the whole process is repeated except MixColumn step.

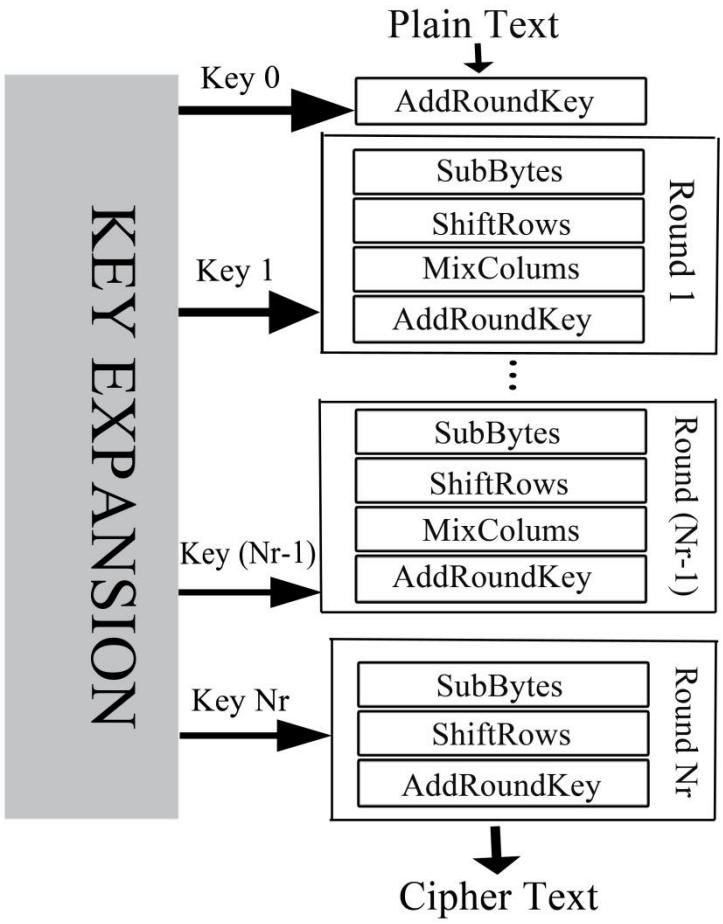

Figure 3.3. Block Diagram of Encryption Rounds

As an example, the message is set to be 8-bit ASCII codes of "HARRANUNIVERSITY", which is “48415252414e554e4956455253495459” in hexadecimal. Then the result is used as plaintext input as shown in Table 3.8.

Table 3.8. Plain text (128-bit)

48	41	52	52	41	4e	55	4e	49	56	45	52	53	49	54	59

The first 4 bytes of the Plain text became the first column of the message state and the second 4 bytes become the second column and so on as shown below in Table 3.9.

Table 3.9. Message state

48	41	49	53
41	4E	56	49
52	55	45	54
52	4E	52	59

3.1.1.6. Add Round Key

In initial round, the plain text in Message State form is XORed with a 128-bit round key in Key State form (Key 0) as Add Round Key operation in order to generate a new state. The Add Round Key is a step in all remaining rounds. For each round, a sub key is extracted from the main key using Rijndael key schedule; each sub key is the same size as the state. The Add Round key operation is implemented by bitwise-XORing each byte of the state with the corresponding byte of the sub key as shown below in Table 3.10.

Table 3.10. Add Round Key

48	41	49	53	XOR	AB	34	52	BA	=	E3	75	1B	E9
41	4E	56	49		12	92	47	63		53	DC	11	2A
52	55	45	54		C3	E8	AD	E8		91	BD	E8	BC
52	4E	52	59		D0	7F	86	1F		82	31	D4	46
Message state					Key 0					Add Round Key			

3.1.1.7. Sub Bytes

A basic substitution of each state byte is realized as in Key Substitution by using 16x16 bytes S-Box given in Table 3.5. Each state byte is transformed into two hexadecimal digits as XY in which the first 4 bits is X and the last 4 bits is Y. Sub Bytes are created by replacing each state byte with the value in S-Box indexed with X and Y. As an example, the very first 8 bits of the state contains E3, where X = E and Y = 3. The intersection of Row “E” and Column “3” in S-Box Table has “11”. The result table is presented in Table 3.11.

Table 3.11. Sub Bytes

E3	75	1B	E9	→	11	9D	AF	1E
53	DC	11	2A		ED	86	82	E5
91	BD	E8	BC		81	7A	9B	65
82	31	D4	46		13	C7	48	5A
Add Round Key State					Sub Bytes State			

3.1.1.8. Shift Rows

In Shift Rows step, bytes are shifted in rows. The first row stays unchanged, the second row is shifted 1 Byte to the left, and the third row is shifted 2 Bytes to the

left and the fourth row is shifted 3 Bytes to the left. Then the result will be saved in a new state as shown below in Table 3.12.

Table 3.12. Shift Rows

11	9D	AF	1E
ED	86	82	E5
81	7A	9B	65
13	C7	48	5A

Sub Bytes State

→

11	9D	AF	1E
86	82	E5	ED
9B	65	81	7A
5A	13	C7	48

Shift Rows State

3.1.1.9. Mix Columns

The Mix Columns is utilized by the Rijndael algorithm after the ShiftRows step. This combination can provide diffusion property in the Rijndael cryptography. The Mix Columns operation is employed in all following rounds except the final round. Each column of Shift Row states is viewed as a four-part polynomial in the finite Galois Field-GF(2^8) and multiplied with a 4x4 constant matrix in modulo (x^4 + 1) as shown in Table 3.13. This constant matrix is called as Maximum Distance Separable (MSD) matrix possessing certain diffusion properties. While multiplication in finite field requires a division by modulo value, addition can be done with XOR operation. As an example, the first byte of the Matrix Multiplication can be calculated as follows: 02*11⊕ 03*86⊕ 01*9B⊕ 01*5A= 72.

Table 3.13. Mix Columns

02	03	01	01
01	02	03	01
01	01	02	03
03	01	01	02

MSD Constant Matrix

×

11	9D	AF	1E
86	82	E5	ED
9B	65	81	7A
5A	13	C7	48

Shift Rows State

→

72	C	37	22
EA	3E	21	19
54	0	01	DF
9A	7D	1B	25

Mix Columns State

3.1.1.10. Add Round Key

Add Round Key operation as a part of rounds will be performed in each round. The previous result State is XORed with a corresponding round key as shown below in Table 3.14. The resulting state will be an input state to the next round.

Table 3.14. Add Round Key

72	CA	37	22	⊕	51	65	37	8D	=	23	AF	00	AF
EA	3E	21	19		89	1B	5C	3F		63	25	7D	26
54	E0	01	DF		03	EB	46	AE		57	0B	47	71
9A	7D	1B	25		24	5B	DD	C2		BE	26	C6	E7
Mix Columns State					Round Subkey					Round Result State			

The results of rounds in AES encryption from the plain text to the cipher text are given below. For 128-bit key, 10 intermediate results are produced.

Round 1: 236357BEAF250B26007D47C6AF2671E7

Round 2: 1F5AC3BFA6BAFFB21CA6AB0382483615

Round 3: DC9E1A9F114FDE9F80738C90272E6E6B

Round 4: 47BE29BA353AF0D7035B3CB8FE54D653

Round 5: D7B5D7C08B08E7FB600A49B8241D784A

Round 6: E7670C36661D710DA3DEB58D5E0E72F7

Round 7: 531E047A9D920180AC97482E03B61D90

Round 8: C2EAF2B8382904AADA82F48EFA39401E

Round 9: F54B31D6A19004474CD68DB605B6B258

Round 10: EEBC5ADAC5EC86CA2FDD580B4BBAA18D

The output of AES encryption algorithm is generated at Round 10 and the encrypted Cipher text for “HARRANUNIVERSITY” in hexadecimal is:

EEBC5ADAC5EC86CA2FDD580B4BBAA18D.

3.1.2. Decryption Block for AES Algorithm

Decryption is the inverse of Encoding method. Encrypted information can be translated into the original message again in AES algorithm. Encoding and Decoding operations are not the same, but the main steps for each are similar. Decryption has four steps of transition, which are Add Round Key, Inverse Shift Rows, Inverse Substitute Bytes and Inverse Mix Columns. Decryption of the cipher text with a 128-bit key passes through one initial round and 10 more rounds as shown in Figure 3.4. The initial round starts with the cipher text and the last round Key from the Encryption Block and performs an Add Round Key operation. The following rounds go through Inverse Shift Rows, Inverse Substitute Bytes, Add Round Key and Inverse Mix Columns operations, but there is no Inverse Mix Columns in the final round.

3.1.2.1. Inverse Rounds

Decoding a cipher text in Rijndael cryptography proceeds through one initial round of Add Round key operation and 10 rounds of Inverse Shift Rows, Inverse Substitute Bytes with an Inverse S-Box, Add Round Key with a proper Key and Inverse Mix Columns operations as seen in Figure 3.4.

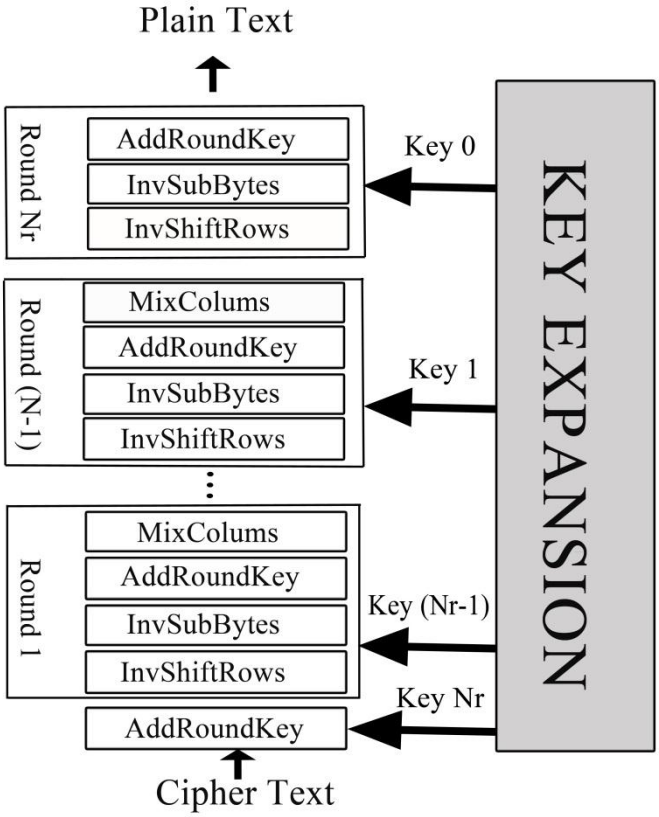

Figure 3.4. Block Diagram of Decryption Inverse Rounds

The output of the Encryption block is the Cipher text (128-bit) as follows: EEBC5ADAC5EC86CA2FDD580B4BBAA18D.

Table 3.15. Cipher Text (128-bit)

EE	BC	5A	DA	C5	EC	86	CA	2F	DD	58	0B	4B	BA	A1	8D

Table 3.16. Cipher state

EE	C5	2F	4B
BC	EC	DD	BA
5A	86	58	A1
DA	CA	0B	8D

3.1.2.2. Add Round Key

Each byte of the Cipher state is XORed with the corresponding byte of Round Key state as shown in Table 3.17.

Table 3.17. Add Rounds key

EE	C5	2F	4B	XOR	08	F7	06	20	=	E6	32	29	6B
BC	EC	DD	BA		DC	1A	93	09		60	F6	4E	B3
5A	86	58	A1		07	B1	9F	53		5D	37	C7	F2
DA	CA	0B	8D		B0	3C	AB	C3		6A	F6	A0	4E
Cipher State					Round Key 10					Initial Round State			

3.1.2.3. Inverse Shift Rows

The Inverse Shift Rows operation is the opposite of Shift Rows and the rows of the state are shifted in the opposite direction. The inverse shift rows state is formed by keeping the first row unchanged, the second row shifted one byte to the right, the third-row shifted two bytes to the right, and the fourth-row shifted three bytes to the right as shown in Table 3.18.

Table 3.18. Inverse Shift Rows

E6	32	29	6B	→	E6	32	29	6B
60	F6	4E	B3		B3	60	F6	4E
5D	37	C7	F2		C7	F2	5D	37
6A	F6	A0	4E		F6	A0	4E	6A
Round State					Inverse Shift Rows			

3.1.2.4. Inverse Sub Bytes

The transferred data from the Inverse Shift Row is translated into the new state using the Inverse S-Box. The inverse S-Box is shown in Table 3.19 and the inverse Sub Bytes state is formed as in Table 3.20.

Table 3.19. Inverse S-Box

	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***A***	***B***	***C***	***D***	***E***	***F***
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Table 3.20. Inverse Sub Bytes

E6	32	29	6B
B3	60	F6	4E
C7	F2	5D	37
F6	A0	4E	6A

Inverse Shift Rows

→

F5	A1	4C	05
4B	90	D6	B6
31	04	8D	B2
D6	47	B6	58

Inverse Sub Bytes

3.1.2.5. Add Round Key

At the Add Round Key step, Inverse Sub Bytes are XORed with a proper Round Key, which is Key 9 in the first round. Add Round State will be formed as shown in Table 3.21.

Table 3.21 Add Round Key

F5	A1	4C	05	⊕	86	FF	F1	26	=	73	5E	BD	23
4B	90	D6	B6		97	C6	89	9A		DC	56	5F	2C
31	04	8D	B2		42	B6	2E	CC		73	B2	A3	7E
D6	47	B6	58		47	8C	97	68		91	CB	21	30

Inverse Sub Bytes — Round Key 9 — Add Round Key State

3.1.2.6. Inverse Mix Columns

The Inverse Mix Columns operation is employed in all following rounds except the final round. Each column of Shift Row states is viewed as a four-part polynomial in the finite Galois Field-GF(2^8) and multiplied with a 4x4 inverse constant matrix in modulo (x4 + 1) as shown in Table 3.22. This constant matrix is the Inverse of MSD matrix given in the encryption block.

Table 3.22. Inverse Mix Columns

0E	0B	0D	09	✖	73	5E	BD	23	→	25	07	57	2D
09	0E	0B	0D		DC	56	5F	2C		A5	13	12	87
0D	09	0E	0B		73	B2	A3	7E		BF	09	89	F2
0B	0D	09	0E		91	CB	21	30		72	6C	AC	19

Inverse Constant Matrix — Add Round Key — Inverse MixColumns

AES Decryption of the Cipher text to the Plain text implements one initial round and ten more rounds containing the previously explained steps. The intermediate results of the rounds are presented below.

Inverse Round 1: 25A5BF720713096C571289AC2D87F219

Inverse Round 2: ED4F52605E88A4DA914EF2CD7B727C31

Inverse Round 3: 94A4D568331D40050AABFED75885A35D

Inverse Round 4: 0E303BD63D67BCBAD0A40E0F36D5946C

Inverse Round 5: A080EBED9639F6F47B20A50EBBAE8C6C

Inverse Round 6: 8684647F828F9FDBCD31A2DBCC0B1D60

Inverse Round 7: C0F46259242405089C522E3713BE167B

Inverse Round 8: 263FA09479FFA3AE63F75BF779FB2BB4

Inverse Round 9: 11869B5A9D826513AFE581C71EED7A48

Inverse Round 10: 48415252414E554E4956455253495459

The output of AES decryption algorithm, called as plain text, is ASCII Codes for "HARRANUNIVERSITY", and is shown in hexadecimal in Table 3.23.

Table 3.23. Plain text (128-bit)

48	41	52	52	41	4E	55	4E	49	56	45	52	53	49	54	59

3.2. Data Encryption Standard (DES)

DES (Data Encryption Standard) algorithm is another symmetric cipher algorithm of network security. The DES algorithm is developed in the early 1970s at IBM and submitted to the interested groups to check the strength and the security of the DES algorithm, DES is one of the landmarks in cryptographic algorithms. DES uses block cipher technique for encryption and decryption (Padmapriya and Subhasri, 2013; Seth and Mishra, 2011). The fundamental concept of DES algorithm is shown in Figure 3.5.

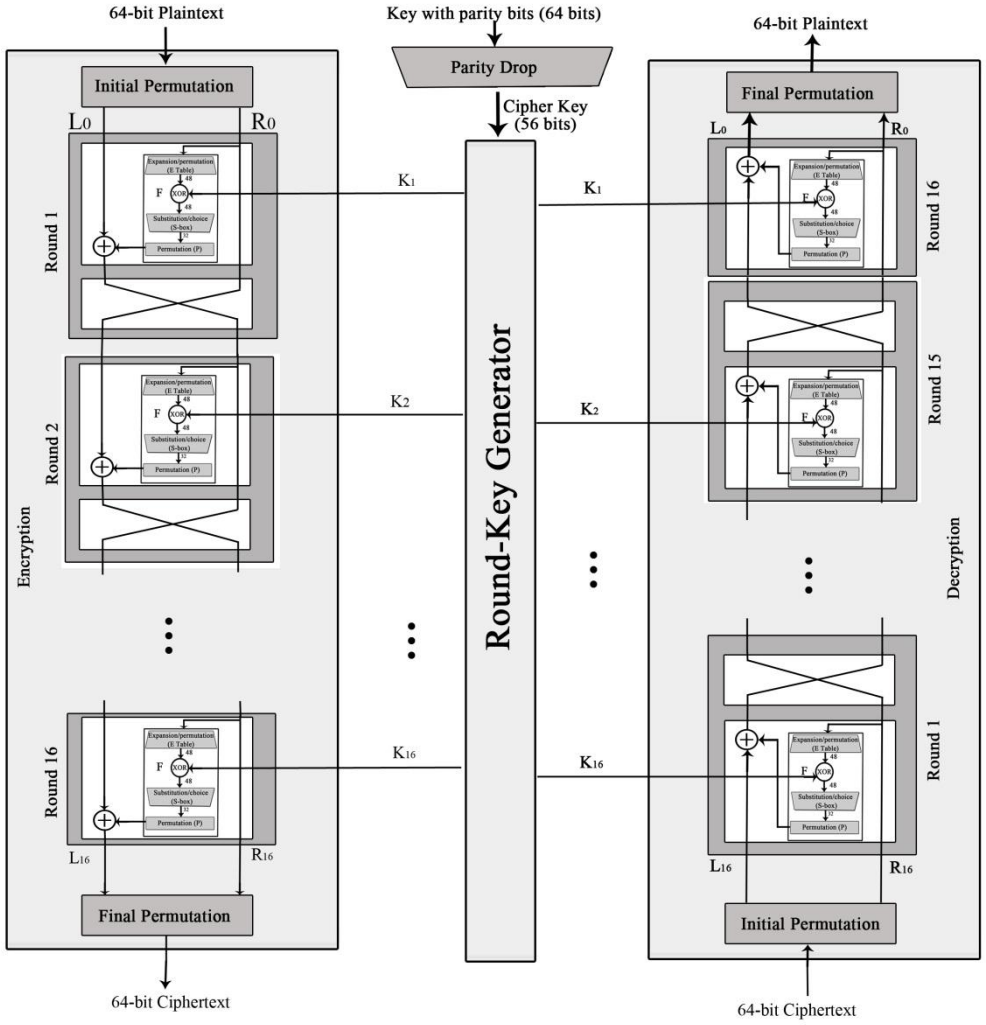

Figure 3.5. Overview of DES Algorithm

DES is a block cipher that encrypts data in 64-bit blocks. DES accepts 64 bits of plaintext as an input and produces 64-bit ciphertext as output. For encryption and decryption, with slight changes, the same algorithm and the same key are used. Despite receiving 64-bit key, the actual key length is 56 bits. The remaining 8 bits are used for error correction.

3.2.1. Encryption Block for DES Algorithm

64-bit plain text is first applied to the initial permutation and 64-bit permutated output is produced according to the predefined rules. The output of the initial permutation is divided into 32-bit left block and 32-bit right block. Both blocks go through 16 rounds of encryption with an appropriate 48-bit key for each round. Round-key generator produces 16 keys using 56 bits of the original 64-bit key. The encryption rounds output 32-bit Left and 32-bit Right blocks. These blocks are united after a swap operation. Final permutation which is the inverse of the initial permutation is performed on the combined block and 64-bit cipher text block is generated as shown in Figure 3.6 (Alioto et al., 2010).

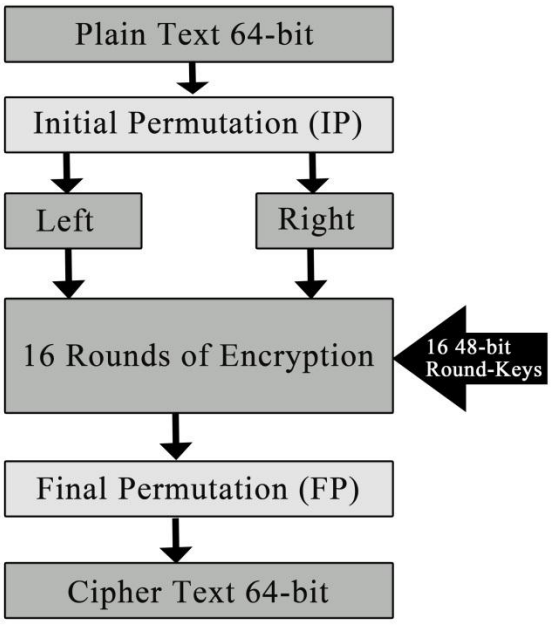

Figure 3.6. Block Diagram of DES Encryption

3.2.1.1. Initial Permutation and Final Permutation

The initial permutation (IP) accepts 64-bit plain text as input and generates 32-bit left block and 32-bit right block by rearranging bit positions as shown in Figure 3.7. according to predefined rules. IP is a keyless and reversible process. After the permutated left and right blocks are processed by 16 rounds of encryption, the combined block of 64 bits is applied to the final permutation (FP) in order to inverse the effect of IP. FP is also a keyless process of bit position arrangements as shown in Figure 3.7. FP generates 64-bit cipher text. The permutation rules for IP and FP are given in Table 3.24. and Table 3.25, respectively.

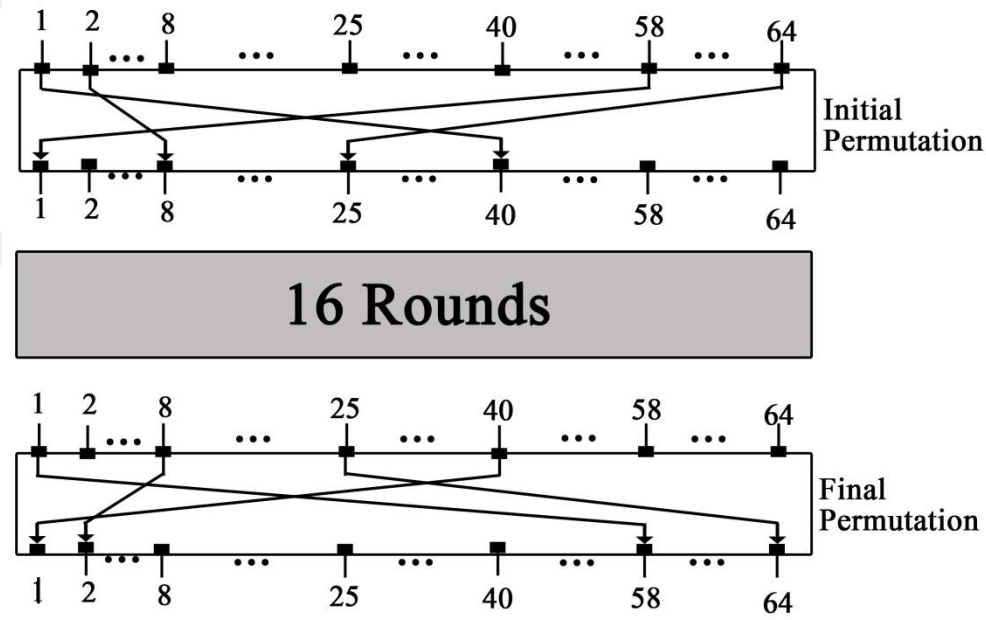

Figure 3.7. Initial Permutation(IP) and Final Permutation(FP)

Table 3.24. Initial Permutation(IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Table 3.25. Final Permutation(FP): Inverse IP

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

3.2.1.2. Round-Key Generation

As mentioned before, each round is processed with an appropriate 48-bit key. Round-key generator produces these 16 keys using 56 bits of the original 64-bit key. DES key consists of 56-bit key and 8-bit parity. Parity bits are 8th bit of each byte starting from the left. To drop 8-bit parity and to permute the remaining bits, all bits are connected to the output according to Permuted choice PC-1 as seen in Table 3.26. The resulting 56-bit key is divided into two parts equally: the left(C_0) and the right key(D_0). New keys can be generated by rotating the previous key left according to the rules given in Figure 3.8. For example, to produce C_1 and D_1, C_0 and D_0 are rotated one bit left. To produce C_2 and D_2, C_1 and D_1 are rotated one bit left again and so on. Shift operation is repeated in each round. For the rounds of 1,2,9 and 16, one bit shift is required, but for the rest, two-bit shift is required. 48-bit round-keys are generated by connecting appropriate bits of Ci and Di keys for each round to output according to Permuted Choice PC-2 as shown in Table 3.27.

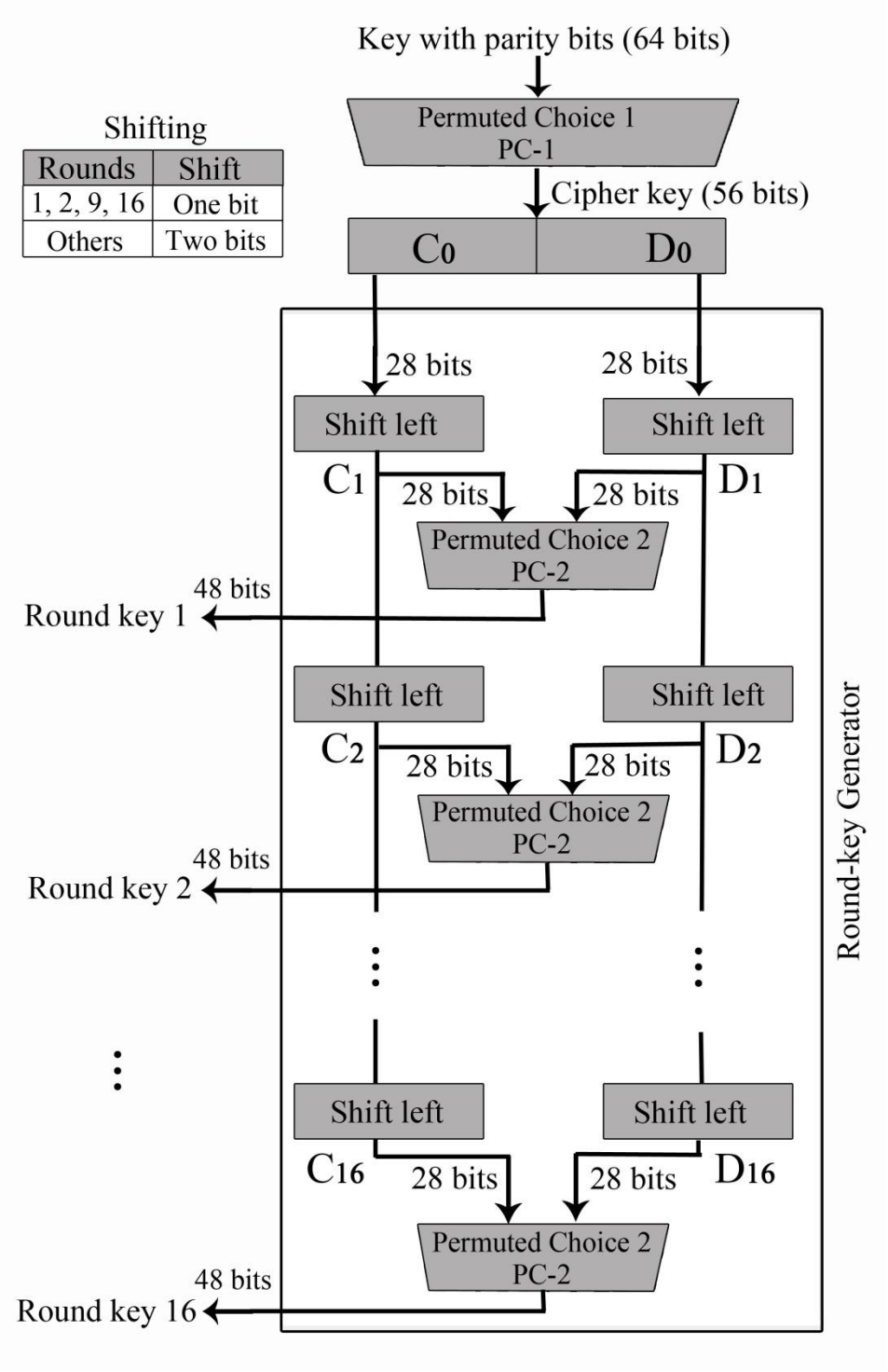

Figure 3.8. .Key Generator of DES Algorithm

Table 3.26. Permuted Choice PC-1

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Table 3.27. Permuted Choice PC-2

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

As an example, DES 64-bit key is selected as “ABCDEF1123456789” in hexadecimal and given in Table 3.28. According to the round key generator algorithm, the following round keys of K1, K2,…,K16 are calculated and listed below.

Table 3.28. Key (64-bit)

AB	CD	EF	11	23	45	67	89

K1 C57C12F0AC1A

K2 04436E93958D

K3 A251358A17A1

K4 8D0B615A6B25

K5 8372B9724998

K6 9D17C0C1311B

K7 525AC9E73228

K8 19F144701B6E

K9 972C1C5EC504

K10 4E26908865CC

K11 5E9C2CE8F281

K12 CAA04AF2462B

K13 28CE2E9E1B0A

K14 E0390A947370

K15 20AE7171AA60

K16 B8D05442C2DD

3.2.1.3.Rounds of Encryption in DES

The sixteen rounds of encryption start with 32-bit left and 32-bit right blocks and 48-bit key for each round. All rounds contain a function unit (F), XOR and SWAP operations except the last round. Round 16 has a function unit (F) and XOR, but no SWAP operations. Function F receives 32-bit Right block and 48-bit round-key and produces 32-bit output. For 32-bit input to be XORed with 48-bit round key, the Right block requires Expansion Permutation. After XOR operation, the size of the result is compressed to 32 bits by implementing Substitution Permutation with the use of S-boxes. The last step of Function F is another bit-shuffling operation according to permutation box (P-box). The output of Function F is XORed with the input left block. Final operation of each round is to swap the resulting 32-bit left block with the 32-bit right block. Therefore, new left and right blocks are generated at each round (Mandal, 2012) as shown in Figure 3.9. After sixteen rounds of encryption, the result is applied to Final Permutation as explained earlier.

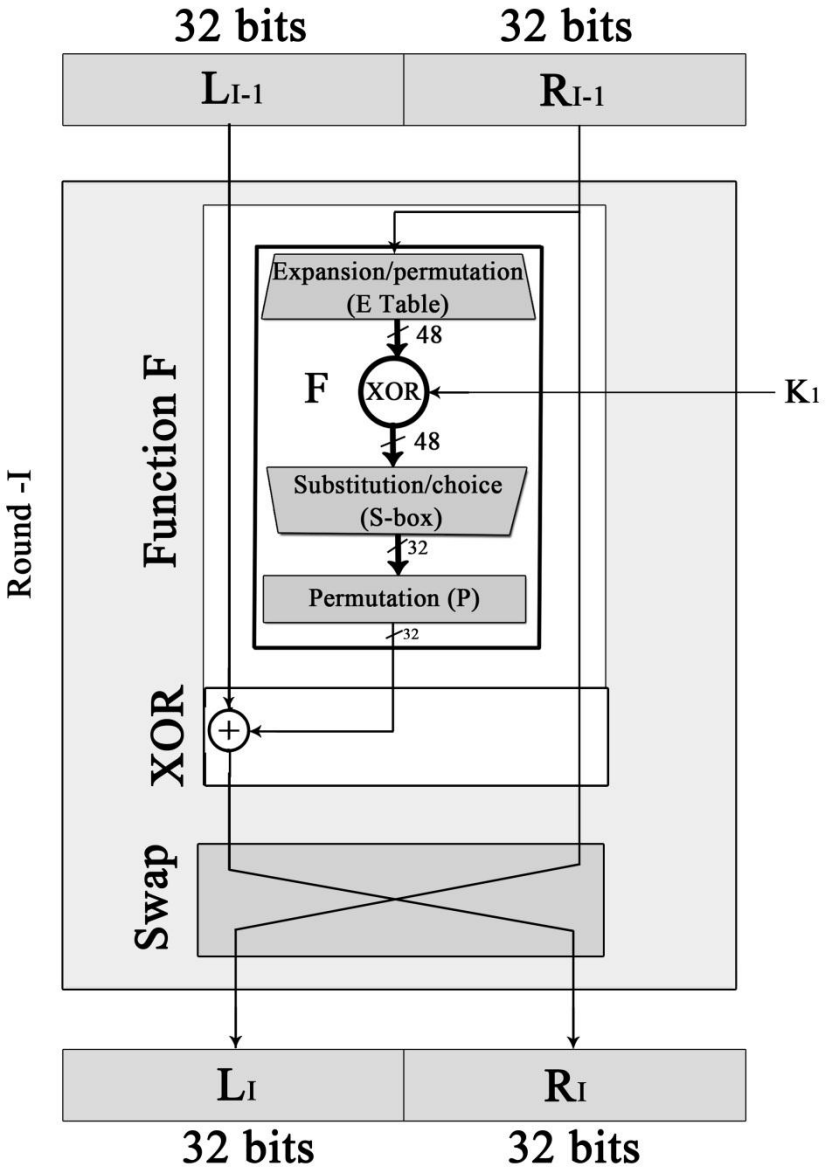

Figure 3.9. Single Round of Encryption

3.2.1.4. Expansion Permutation

Since the right input is 32 bits and the round key is 48 bits, the size of right input must be extended to 48-bits. There are 8 blocks of 4 bits in 32 bits. By reusing some bits according to Table 3.29., it generates 48-bit output. The connections are implemented as shown in Figure 3.10. The output of the Expansion permutation is XORed with a proper Round-key as seen in Figure 3.9.

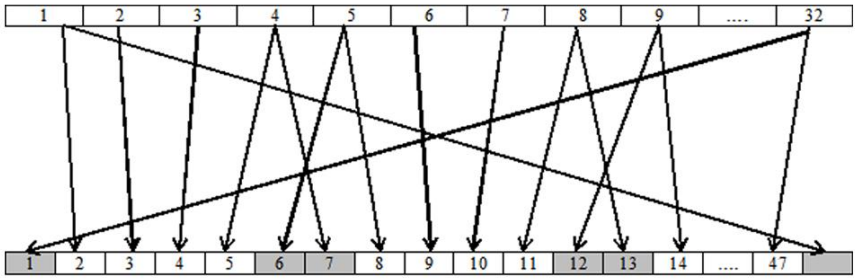

Figure 3.10. Expansion Permutation Connections

Table 3.29. Expansion Permutation(E Table)

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

3.2.1.5. S-Box Substitution

S-Box substitution converts 48-bit input to 32-bit output using eight S-Boxes (S1, S2, S3, S4, S5, S6, S7 and S8) as shown in Table 3.30-3.37. Each S-Box receives 6-bit entry and generates 4-bit output. S-Box tables consists of 4 rows and 16 columns. 48-bit input is divided into eight blocks of 6-bit. Each block starting from the Most Significant Bit(MSB) is applied to S-Box starting from S1. The first and the last bits of 6-bit entry are used to select the row and the middle 4 bits of entry are used to select the column. The corresponding 4-bit output of 6-bit entry is found in S-Box with these row and column numbers. Outputs of eight S-Boxes are then integrated into a 32-bit block.

Table 3.30. S-Box (S1)

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

Table 3.31. S-Box (S2)

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

Table 3.32. S-Box (S3)

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

Table 3.33. S-Box (S4)

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

Table 3.34. S-Box (S5)

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

Table 3.35. S-Box (S6)

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

Table 3.36. S-Box (S7)

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

Table 3.37. S-Box (S8)

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

3.2.1.6. P-Box permutation

A permutation box or (P-box) is a bit-shuffling process used in cryptography to permute or transpose bits, which maintains diffusion and confusion. 32-bit output of S-Box block is applied directly to P-Box permutation with the rules given in Table 3.38.

Table 3.38. Permutation Box

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

3.2.1.7.XOR and Swap

In a round operation, the Left 32-bit block has been left unaffected so far. 32-bit output of P-Box forms the Right block. The Right block is XORed with the left block. The result of XOR operation is swapped with the Right block as shown in Figure 3.9.

As mentioned earlier, after 16 rounds of operation are completed, the results are applied to the Final Permutation which is previously explained in 3.2.1.1. Thus, cipher text is produced.

To show the implementation of DES algorithm, the plain text message is set to 8-bit ASCII codes of "HARRANUN", “48415252414E554E” in hexadecimal, and the 64-bit Key is set to “ABCDEF1123456789” in hexadecimal. In Section 3.2.1.2., Round keys for all rounds were generated using the same 64-bit key and listed there. Here, the plain text is shown in Table 3.39. The results of the rounds are given below as Right and Left blocks.

Table 3.39. Plain text (64-bit)

48	41	52	52	41	4E	55	4E

Right0 Left0: 0000A1ACFF4CE052

Right1 Left1: 382655DD0000A1AC

Right2 Left2: 15C895B1382655DD

Right3 Left3: 9190DF2A15C895B1

Right4 Left4: E0BC05AB9190DF2A

Right5 Left5: 46B5CADEE0BC05AB

Right6 Left6: 748AEB9546B5CADE

Right7 Left7: 3407B9D2748AEB95

Right8 Left8: FD5471643407B9D2

Right9 Left9: C4C4B362FD547164

Right10 Left10: 245A1774C4C4B362

Right11 Left11: 4289759B245A1774

Right12 Left12: C88E3F3E4289759B

Right13 Left13: 9084B79EC88E3F3E

Right14 Left14: A9A0B1CA9084B79E

Right15 Left15: 5BE5A940A9A0B1CA

Right16 Left16: 6204FD875BE5A940

The output of DES encryption algorithm for 8-bit ASCII codes of "HARRANUN", " 48415252414E554E" in hexadecimal, is “ADC1358C846CE62D” in hexadecimal, as called Cipher text shown in Table 3.40.

Table 3.40. Cipher Text (64-bit)

AD	C1	35	8C	84	6C	E6	2D

3.2.2. Decryption Block for DES Algorithm

DES Decryption involves the reverse operations of Encryption algorithm. The cipher text goes through Initial Permutation, 16 rounds with the same 16 round-keys and Final Permutation operations as shown in Figure 3.5. As done in the Encryption block, 64-bit cipher text is transformed into 32-bit left and 32-bit right blocks using the Initial Permutation rules given in Table 3.24. These two 32-bit blocks are applied to the rounds as inputs. There are 16 rounds. However, in the decryption algorithm, Round keys are used in reverse order starting with Round key 16 (K 16). Since each round consists of the same operations as in the Encryption algorithm, they will not be explained in detail again. Round operation starts with Expansion/Permutation converting 32-bit right block into 48-bit output. After XORing 48-bit output with a proper round key of the round, 48-bit result is transformed to 32-bit output by using the Substitution S-Boxes given in Table 3.30-3.37. as explained before. Bits of 32-bit output is more permuted according to P-Box given in Table 3.38. Before going through Swap operation, the permuted result is XORed with the 32-bit left block. The 32-bit right block from previous round is swapped with the result of final XOR and then they form the left and right of the next round. Round 16 does not have Swap operation, and the result is directly connected to Final Permutation. The last 32-bit left and 32-bit right blocks are transformed to 64-bit Plain text by connecting the bits of the blocks to output according to Final Permutation rules given in Table 3.25. Thus, DES Decryption algorithm is completed.

To show the implementation of DES decryption algorithm, the cipher text is set to “ADC1358C846CE62D” in hexadecimal which is obtained from the Encryption of plain text message "HARRANUN" in 8-bit ASCII form, and the 64-bit Key is set to “ABCDEF1123456789” in hexadecimal. In Section 3.2.1.2., Round keys for all rounds were generated using the same 64-bit key given in Table 3.42. and listed there. Here, the cipher text is shown in Table 3.41. The results of the rounds are given below as Right and Left blocks.

Table 3.41. Cipher Text state (64-bit)

AD	C1	35	8C	84	6C	E6	2D

Table 3.42. Key (64-bit)

AB	CD	EF	11	23	45	67	89

Right16 Left16: 6204FD875BE5A940

Right15 Left15: 5BE5A940A9A0B1CA

Right14 Left14: A9A0B1CA9084B79E

Right13 Left13: 9084B79EC88E3F3E

Right12 Left12: C88E3F3E4289759B

Right11 Left11: 4289759B245A1774

Right10 Left10: 245A1774C4C4B362

Right9 Left9: C4C4B362FD547164

Right8 Left8: FD5471643407B9D2

Right7 Left7: 3407B9D2748AEB95

Right6 Left6: 748AEB9546B5CADE

Right5 Left5: 46B5CADEE0BC05AB

Right4 Left4: E0BC05AB9190DF2A

Right3 Left3: 9190DF2A15C895B1

Right2 Left2: 15C895B1382655DD

Right1 Left1: 382655DD0000A1AC

Right0 Left0: 0000A1ACFF4CE052

The output of DES decryption algorithm is 8-bit ASCII Codes for "HARRANUN", “48415252414E554E” in hexadecimal, as called plain text shown in Table 3.43.

Table 3.43. Plain Text (64-bit)

48	41	52	52	41	4E	55	4E

3.3. Overview of FPGA

The Field Programmable Gate Array (FPGA) is an integrated circuit with configurable logic elements and configurable interconnections which can be programmed in the field after fabrication. It is possible to configure the programmable logic elements to replicate the features of specific gates such as AND, OR, XOR, NOT, or more complicated mixture structures such as encoders or simple arithmetic functions. FPGAs also contain storage elements which could be plain flip-flops or more complex storage areas.

Such logical block and interconnections may be configured by the consumer designer after the production phase to enable the FPGA to execute whatever logic operation is necessary. Many supplier produce different types of FPGA chips such as Altera, Xilinx, Lattice Semiconductor, Microchip, Fast Logic, Cypress Semiconductor, Atmel, etc. Among them, Altera and Xilinx are the two well-known FPGA manufacturers with many FPGA system variations. The FPGA system used in this study is ATLYS Circuit Board with FPGA Spartan-6 Family XC6SLX45, as shown in Figure 3.11.

3.3.1. ATLYS Circuit Board with FPGA Spartan-6 Family XC6SLX45

The ATLYS circuit board provides a fully prepared digital circuit design environment with Xilinx Spartan-6 LX45 FPGA. The board has very capable FPGA and peripheral devices such as Ethernet, HDMI port, 128 MB of 16-bit DDR2 memory, USB and audio ports. ATLYS is compliant with Xilinx CAD tools such as the free ISE WebPack software.

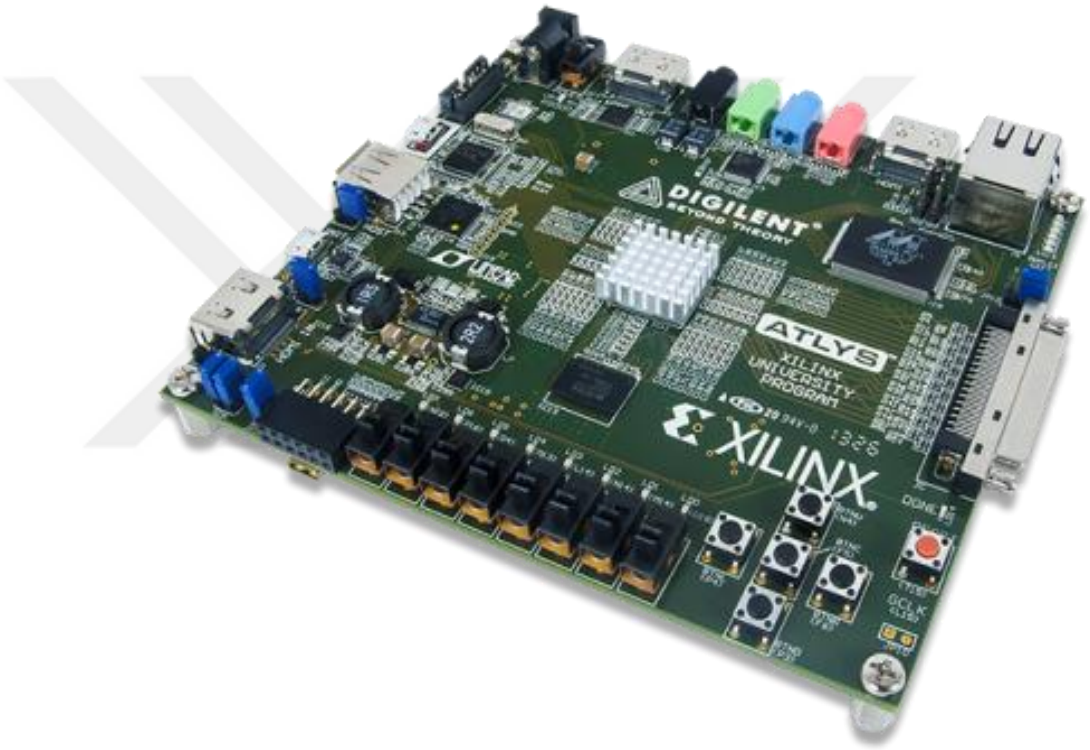

Figure 3.11. FPGA Spartan-6 Family XC6SLX45 Packed CSG324C

The ATLYS board features the new Adept USB2 system from Digilent, which provides real-time power control, online test monitoring, virtual I / O, and user data transfer services. It is shown in Figure 3.12.

ATLYS circuit board provides the following characteristics; Xilinx Spartan-6 LX45 FPGA with 324-pin BGA package, 128MB DDR2, 10/100/1000 Ethernet, USB2 ports for programming and data transfer, USB-UART and USB-HID ports,

two HDMI input ports and two HDMI output ports, 16MB x4 SPI Flash for Configurations and Data Storage, GPIO contains 8 LEDs, 6 buttons, and 8 sliding switches.

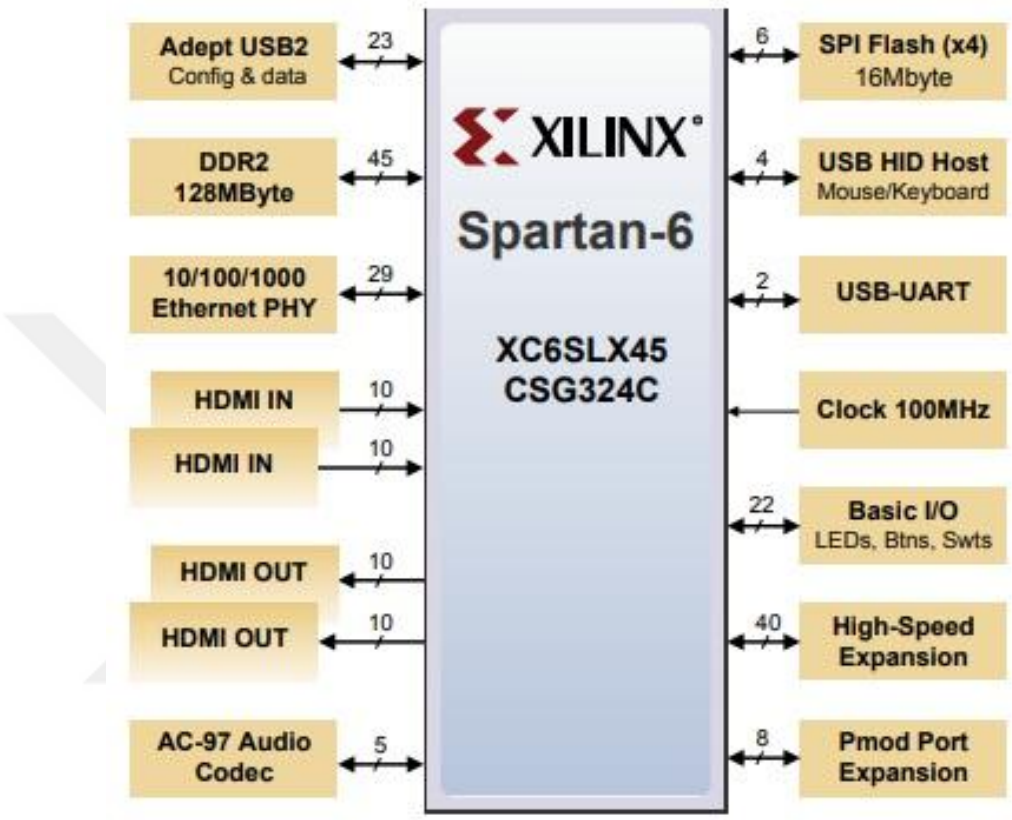

Figure 3.12. ATLYS Circuit Board with FPGA Spartan-6 Family XC6SLX45

3.3.2. Development Tools of Xilinx ISE

Xilinx ISE (Integrated Synthesis Environment) is software platform for synthesizing and evaluating HDL models, allowing developers to preprocess the models, to conduct pacing analyzes, to evaluate RTL graphs, to simulate the design's response to various triggers, and to customize the goal system with the user.

Xilinx ISE is a programming platform of Xilinx FPGA devices, which is directly related to the software of these devices. The Xilinx ISE is mainly used for circuit design and configuration, while ISIM or the ModelSim logic simulation is used for device-level work.

Since 2012, Xilinx ISE has also been withdrawn in favor of the Vivado Design Package, which plays the very same functions as ISE with added device capabilities on a chip creation. Xilinx launched the current iteration of ISE as shown in Figure 3.13. in Oct 2013 (version 14.7), and no further ISE launches are expected. This platform supports Vertex, Spartan, Kintex, Zinc, Artix, XC9500, and Cool Runner series of Xilinx FPGA. It is shown in Figure 3.13.

Logical model simulation of HDL design is performed with ISIM or ModelSim. HDL modeling is a programming-based methodology for accurately modeling the logic design in physical systems. Simulation helps to validate system outputs and timing problems, and to ensure that the application achieves predicted outcomes.

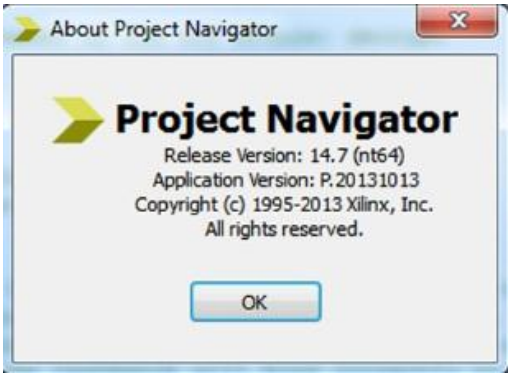

Figure 3.13. Xilinx ISE Design Suite 14.7

3.3.3. Project Development

Cryptography algorithms of AES and DES are developed using Xilinx software that contains the Graphical User Interface (GUI) to access the Spartan 6 FPGA architecture. Xilinx software offers an easy, automatic framework for developers to deliver the highest results in architecture. This program provides the design solutions in VHDL and the design synthesis to maintain reasonable operability and performance.

4. RESULTS and DISCUSSIONS

4.1. Software Implementation

4.1.1. Software Implementation of AES

AES algorithms for encryption and decryption is explained in detail in previous section. Algorithms are divided into blocks such as S-Box Substitution, Shift rows, MixColumn etc. For designing these algorithms, each block is first coded in VHDL as a sub-circuit on the Xilinx ISE platform. Then, AES main design is developed by combining sub-circuits and coding loops of round blocks. The code has been used as the concept for the simulation.

The VHDL code of AES algorithms for encryption and decryption is partially shown in Figure 4.1. and the whole code is given in appendices. In the simulation of AES encryption and decryption algorithms, input and output signals are defined as “encrypt_in128” for encryption input, as “encrypt_out128” for encryption output, as “decrypt_in128” for decryption input and as “decrypt_out128” for decryption output.

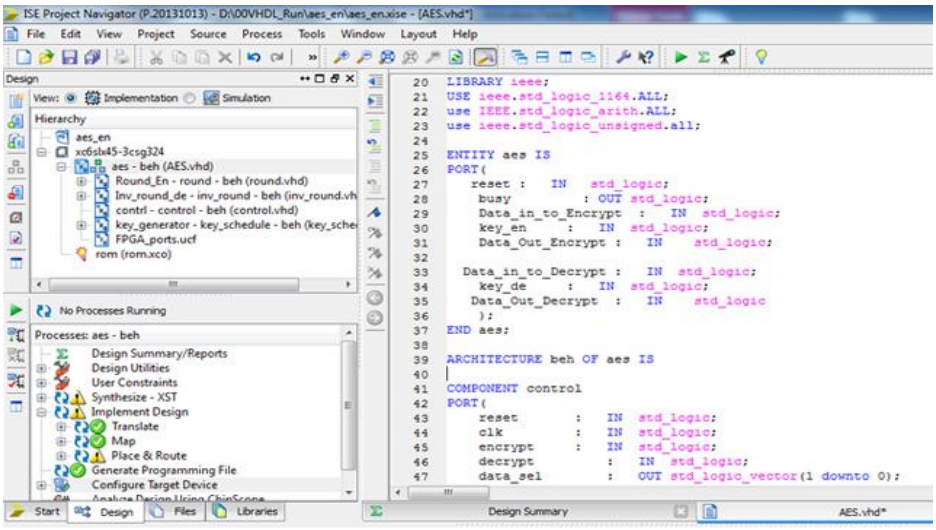

Figure 4.1. Partial Code of AES Algorithms for Encryption/Decryption

Figure 4.2. below shows the RTL block diagram of AES algorithms with its inputs and outputs.

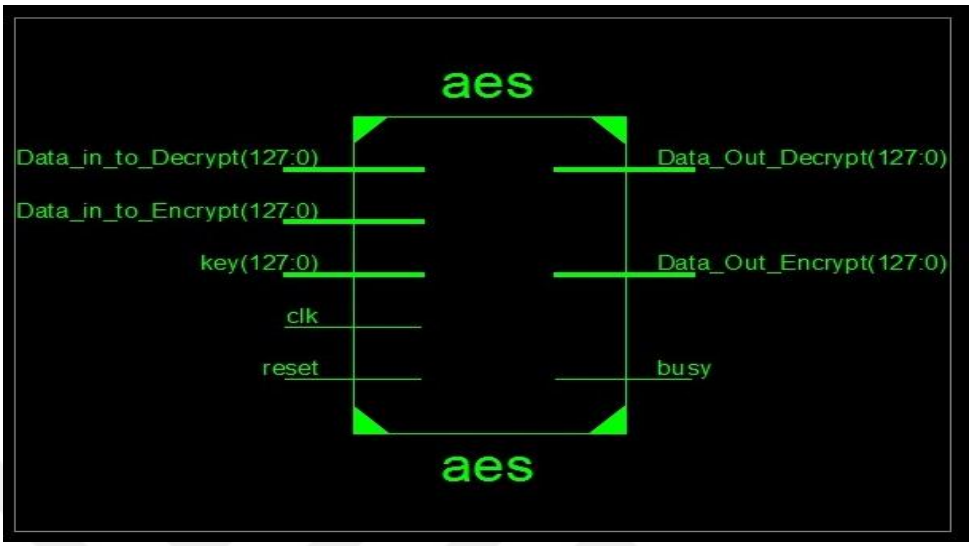

Figure 4.2. RTL Block Diagram of AES Algorithm

Figure 4.3. below shows the partial code of Substitution block with S-Box in AES algorithm.

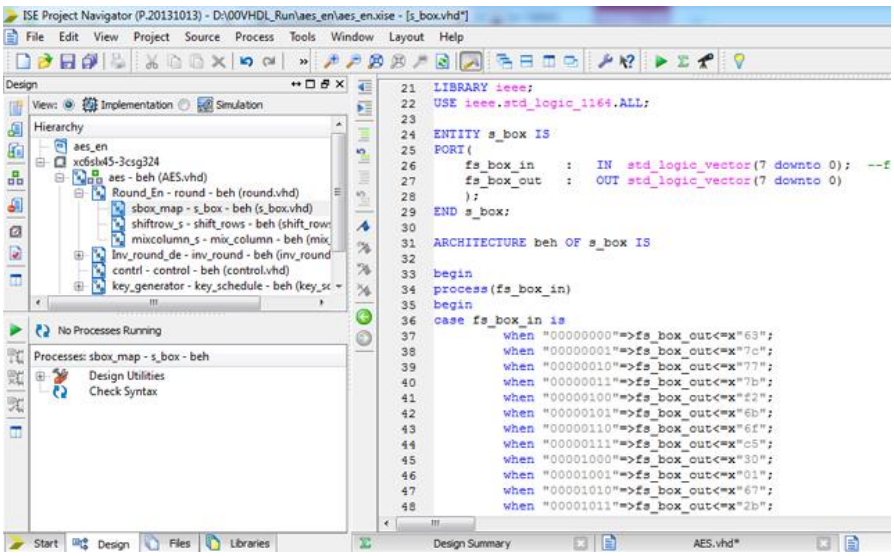

Figure 4.3. Partial Code of Substitution block with S-Box in AES Algorithm

Figure 4.4. below shows the RTL block diagram of Substitution block with S-Box in AES algorithms with its inputs and outputs.

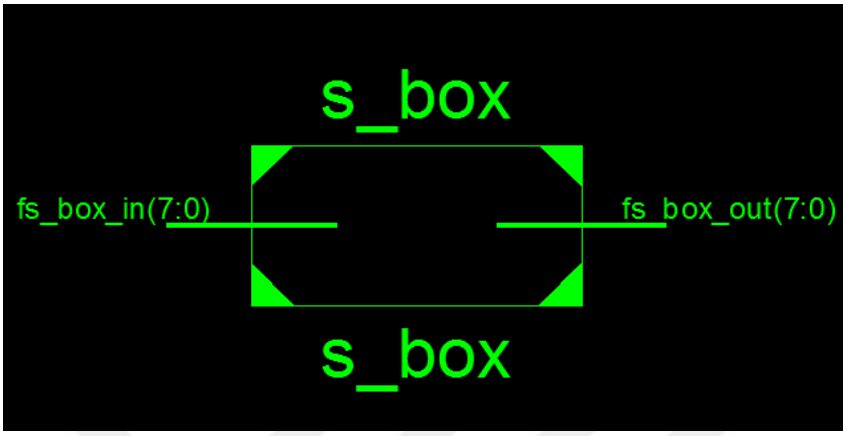

Figure 4.4. RTL Block of S-Box in AES Algorithm

Figure 4.5. below shows the partial code for ShiftRows operation in AES algorithms. This sub-circuit connects each bit of 128-bit input to output bits according to the shift rules given in previous section. Input and Output signals of S-Box are defined as “shiftrow_in” and “shiftrow_out”, respectively.

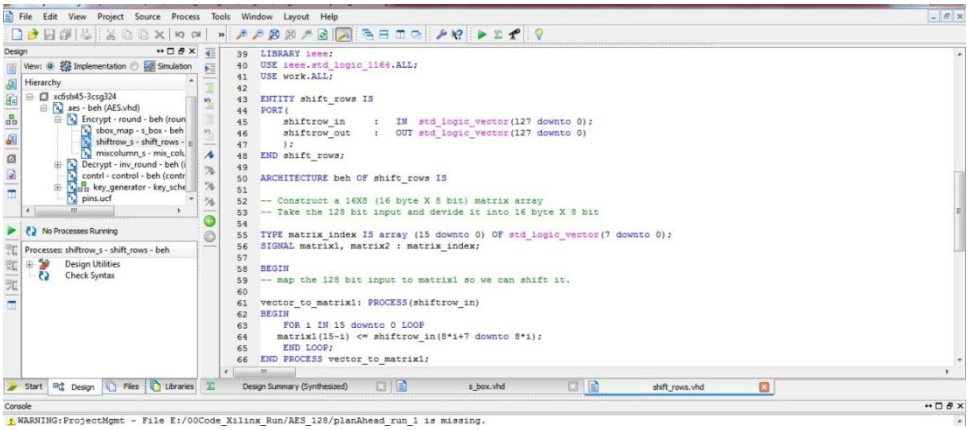

Figure 4.5. Partial Code of Shift Rows in AES Algorithm

Figure 4.6. below shows the RTL block of ShiftRows in AES algorithms with its input and output.

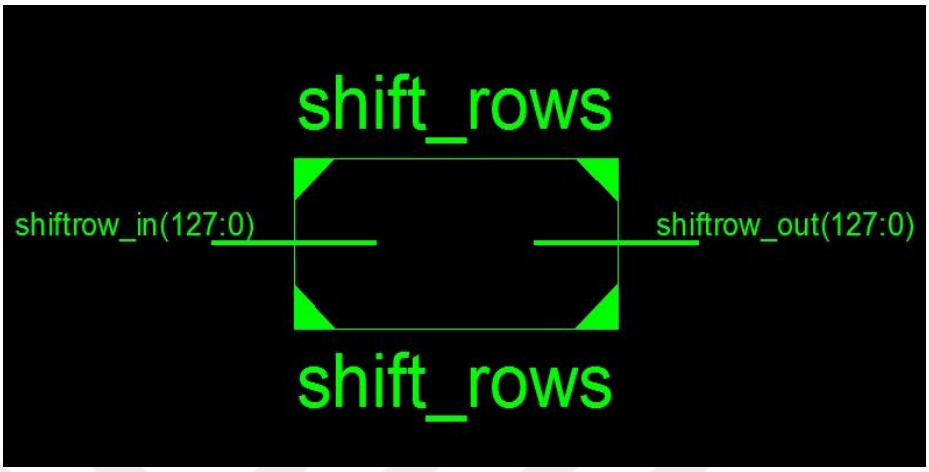

Figure 4.6. RTL Block of ShiftRows in AES Algorithm

Figure 4.7. below shows the partial code of MixColumn in AES algorithms. . Input and Output signals of MixColum sub-circuit in AES algorithm are defined as “mixcolumn_in” and as “mixcolumn_out”, respectively.

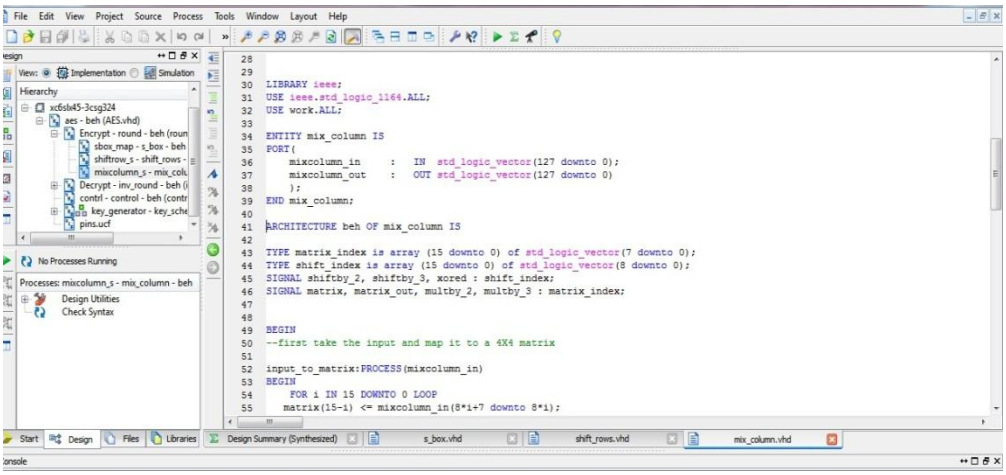

Figure 4.7. Partial Code of Mix Column in AES Algorithm

Figure 4.8. below shows the RTL block of Mix Column in AES algorithm.

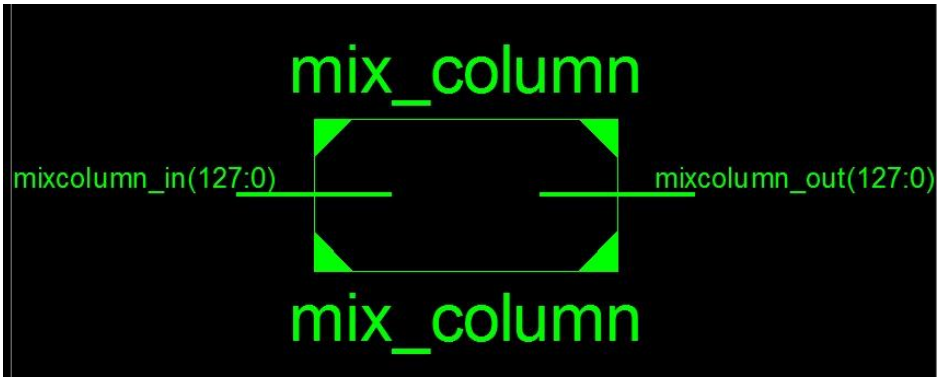

Figure 4.8. RTL Block of Mix Column in AES Algorithm

The implementation of the AES algorithm is simulated on ISIM of the Xilinx ISE platform. The values entered as inputs and the results generated by the simulation of AES algorithm are shown in Figure 4.9. and in Figure 4.10.

For simulation of AES encryption algorithm, 128-bit input key is assigned as "AB12C3D03492E87F5247AD86BA63E81F" in hexadecimal, and the plain text input is assigned as 8-bit ASCII codes of “HARRANUNIVERSITY”, "48415252414E554E4956455253495459" in hexadecimal. The output of AES encryption is generated as "EEBC5ADAC5EC866CA2FDD580B4BBAA18D" in hexadecimal, which is called as cipher text. The resource use of the design is presented in Table 4.1.

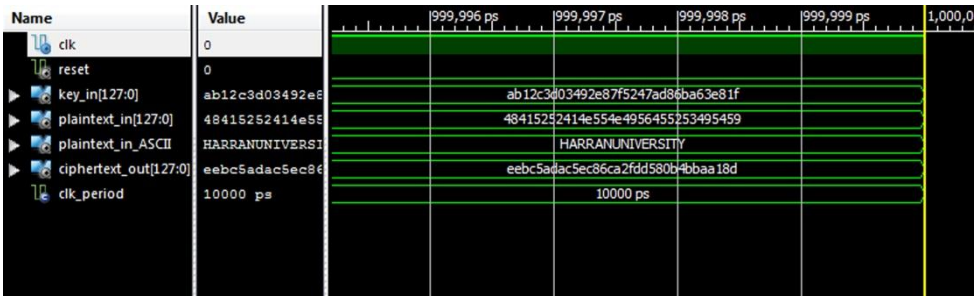

Figure 4.9. Simulation Results of AES Encryption Algorithm

Table 4.1. The resource use of AES Encryption Algorithm

AES Encryption algorithm	***Simulation Results***
Elaboration Time	0.624004 s
Total Signals	64
Total Nets	2099
Total Signal Drivers	60
Total Blocks	38
Total Primitive Blocks	19
Total Processes	57
Total Traceable Variables	25
Total Scalar Nets and Variables	2476
Current Memory Usage	1010.11 MB

Figure 4.10. below shows the Simulation results of AES Decryption Algorithm. AES Decryption Algorithm block is simulated with the same 128-bit key used in the Encryption block and the cipher text input generated in the Encryption block which is "EEBC5ADAC5EC866CA2FDD580B48BAA18D" in hexadecimal. The plain text output is produced as "48415252414e554e4956455253495459" in hexadecimal. The resource use of the design is shown in Table 4.2.

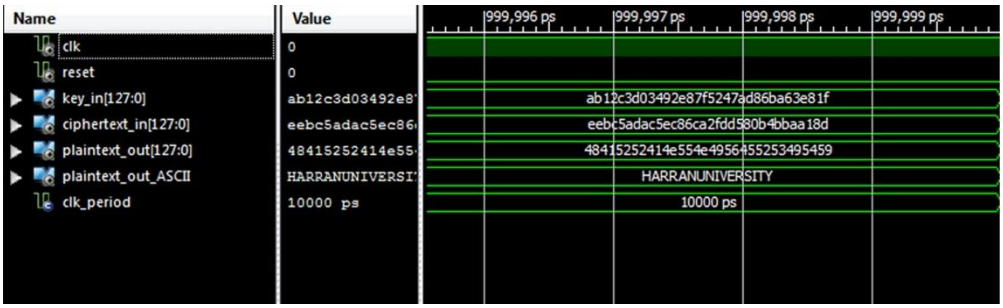

Figure 4.10. Simulation Results of Decryption AES Algorithm

Table 4.2. The resource use of AES Decryption Algorithm

AES Decryption algorithm	***Simulation Results***
Elaboration Time	0.624004 s
Total Signals	64
Total Nets	1924
Total Signal Drivers	60
Total Blocks	38
Total Primitive Blocks	19
Total Processes	60
Total Traceable Variables	25
Total Scalar Nets and Variables	2301
Current Memory Usage	695.542 MB

Table 4.3 shows the complete summary of our implemented design, which describes the resource use for Encryption and Decryption of the AES Algorithm.

Table 4.3. The resource use of Simulation of AES Encryption and Decryption

AES	***Encryption Result AES***	***Decryption Result AES***
Elaboration Time	0.624004 s	0.624004 s
Total Signals	64	64
Total Nets	2099	1924
Total Signal Drivers	60	60
Total Blocks	38	38
Total Primitive Blocks	19	19
Total Processes	57	60
Total Traceable Variables	25	25
Total Scalar Nets and Variables	2476	2301
Current Memory Usage	1010.11 MB	695.542 MB

After simulating the AES algorithm, it can be observed that AES requires 0.624004 seconds for encryption processes and 0.624004 seconds for decryption. Current Memory Usage was 1010.11 MB for encryption and 695.542 MB for decryption.

4.1.2. Software Implementation of DES

DES algorithms for encryption and decryption is explained in detail in previous section. Algorithms are divided into blocks such as Initial Permutation, 16 Rounds, Final Permutation, Round-key Generator, etc. For designing these algorithms, each block is first coded in VHDL as a sub-circuit on the Xilinx ISE platform. Then, DES main design is developed by combining sub-circuits and coding loops of round blocks. The code has been used as the concept for the simulation.

The VHDL code of DES algorithms for encryption and decryption is partially shown in Figure 4.11. and the whole code is given in appendices. In the simulation of DES encryption and decryption algorithms, input and output signals are defined as "Data_in_to_Encrypt" for encryption input, as "Data_Out_Encrypt" for encryption output, as "Data_in_to_Decrypt" for decryption input and as "Data_Out_Decrypt" for decryption output.

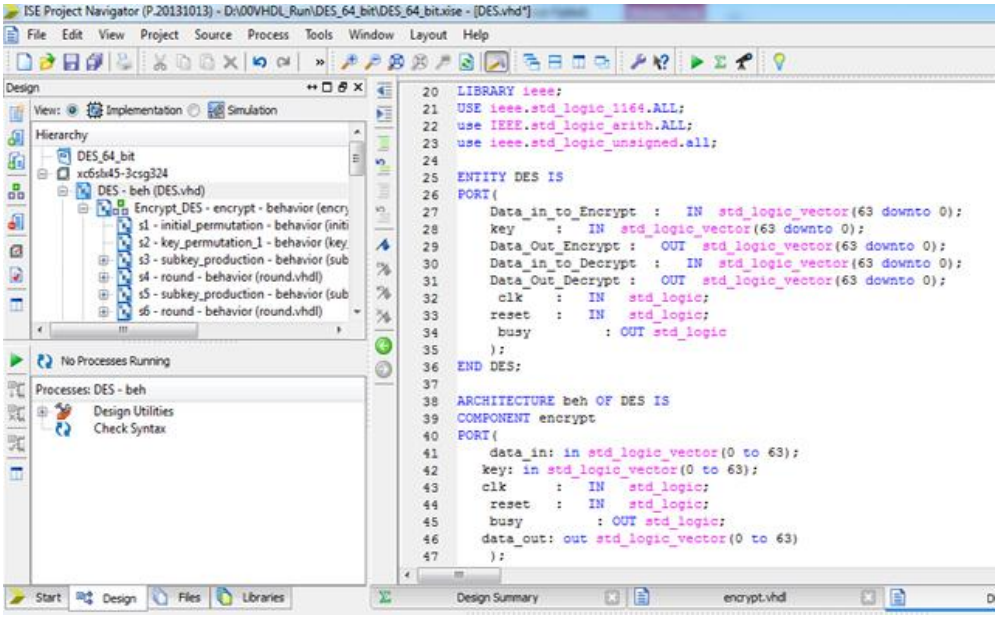

Figure 4.11. Partial Code of DES Algorithm

Figure 4.12. below shows the RTL block diagram of DES algorithms with its inputs and outputs.

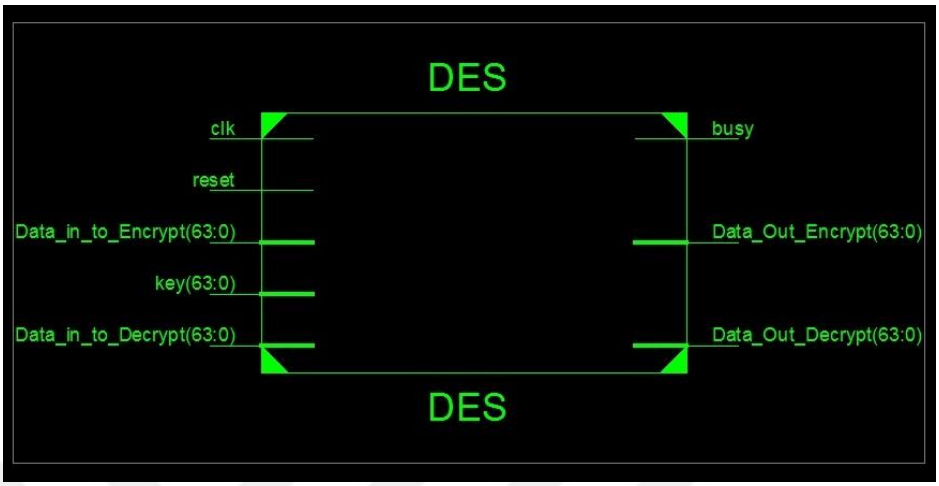

Figure 4.12. RTL Block Diagram of DES Algorithm

Figure 4.13. below shows the partial code of Round Key Generator in DES algorithms.

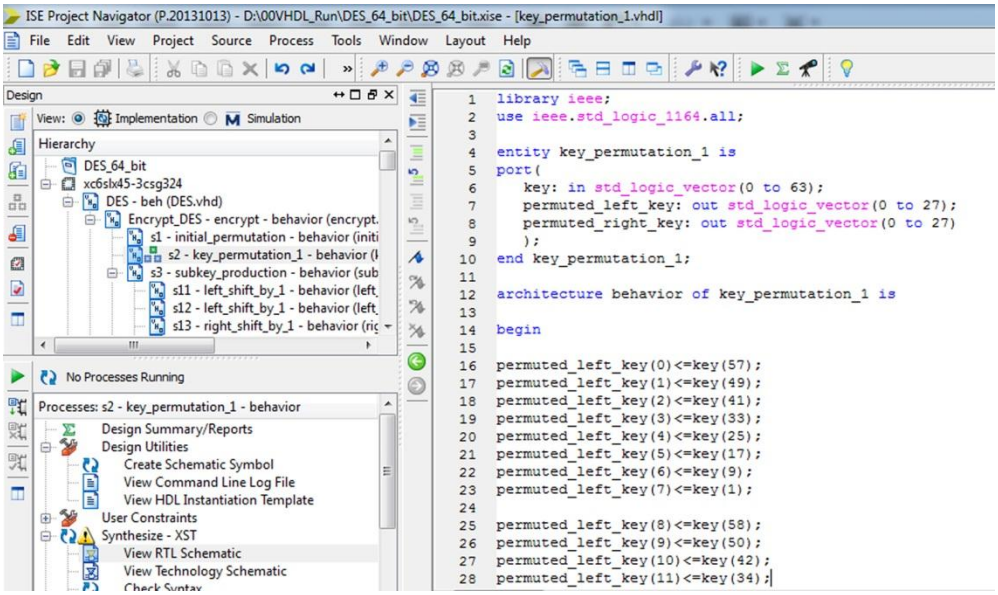

Figure 4.13. Partial Code of Round Key Generator in DES Algorithm

Figure 4.14 below shows the RTL Block of Round Key Generator in DES algorithms with its inputs and outputs.

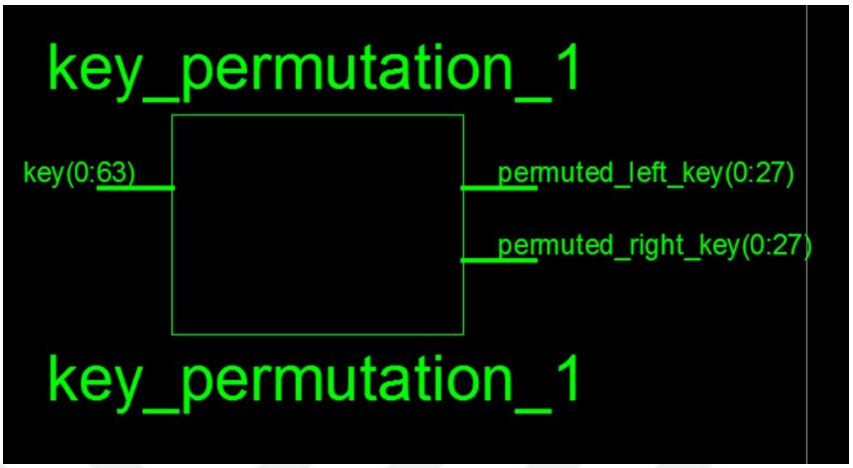

Figure 4.14. RTL Block of Round Key Generator in DES Algorithm

Figure 4.15. below shows the partial code of Round in DES algorithms.

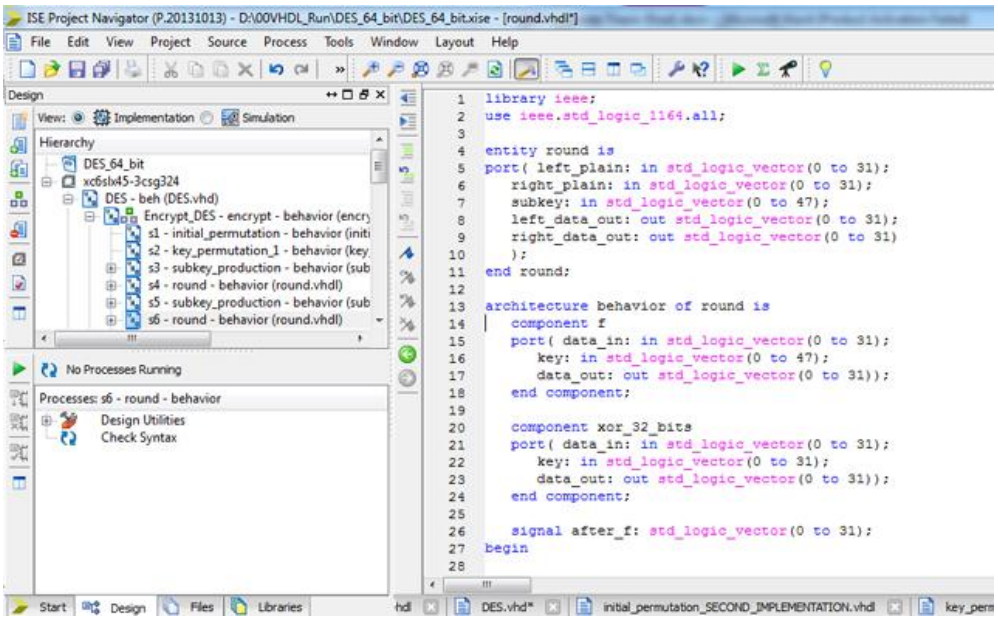

Figure 4.15. Partial Code of Round in DES Algorithm

Figure 4.16. RTL Block of Round in DES Algorithms

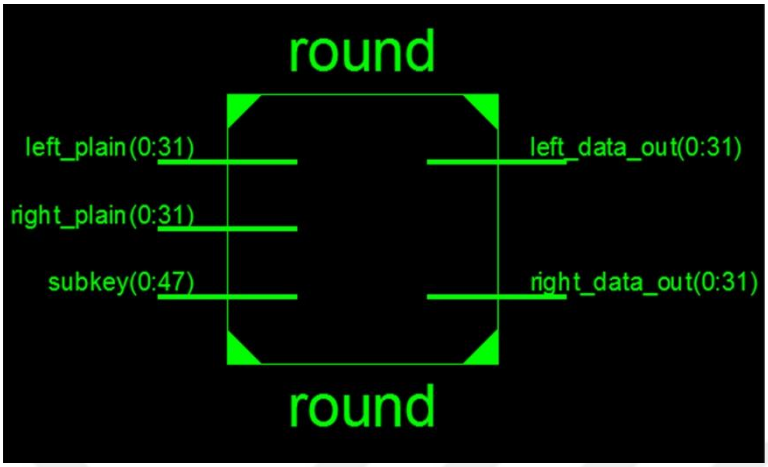

Figure 4.16. RTL Block of Round in DES Algorithm

Figure 4.17. below shows the partial code of Initial Permutation in DES algorithms.

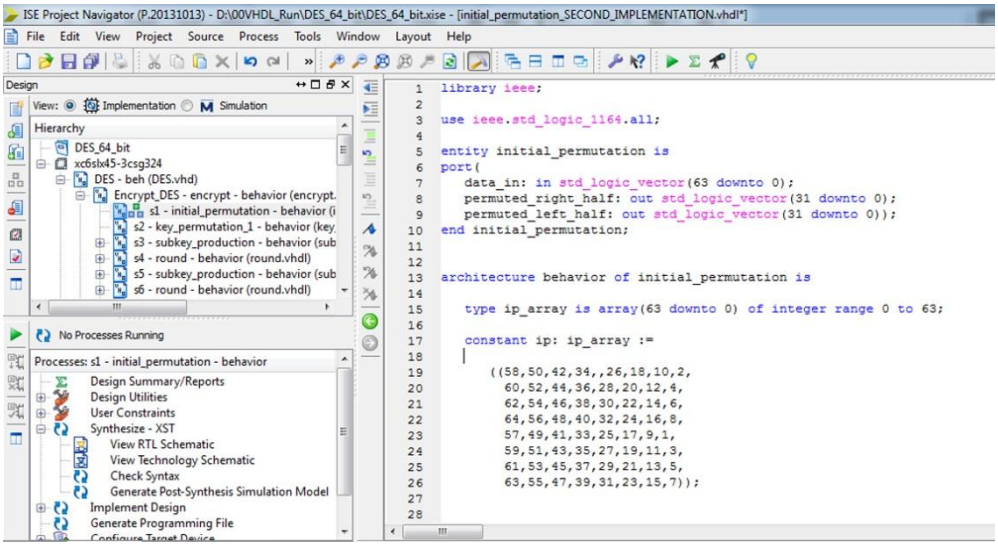

Figure 4.17. Partial Code of Initial Permutation in DES Algorithm

Figure 4.18 below shows the RTL Block of Initial Permutation in DES algorithms with its inputs and outputs.

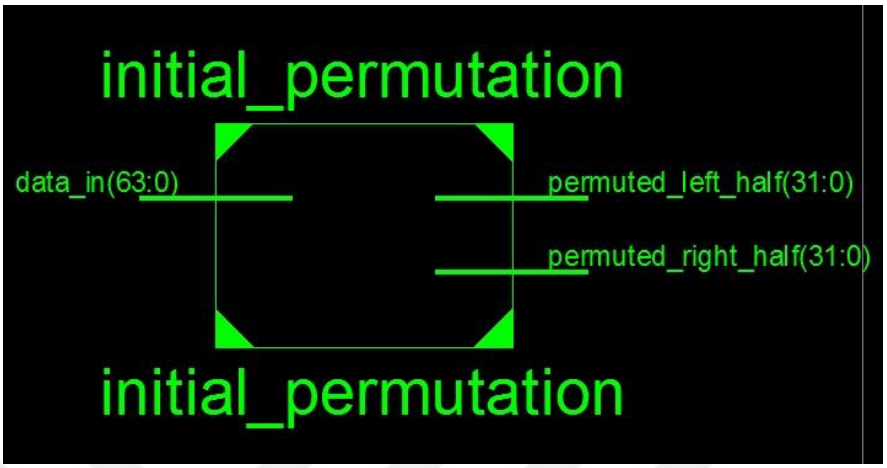

Figure 4.18. RTL Block of Initial Permutation in DES Algorithm

Figure 4.19. below shows the partial code of Final Permutation in DES algorithms.

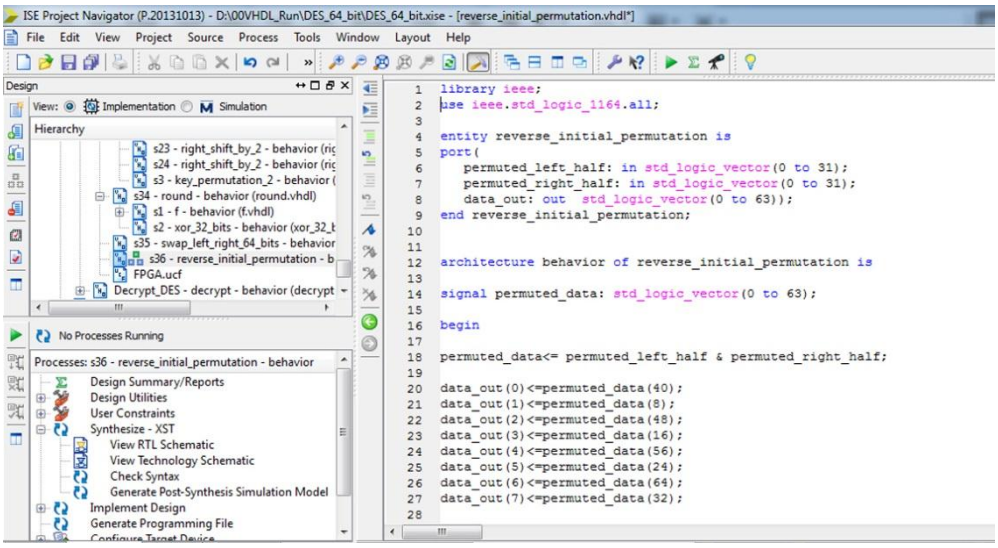

Figure 4.19. Partial Code of Final Permutation in DES Algorithm

Figure 4.20 below shows the RTL Block of Final Permutation in DES algorithms with its inputs and outputs.

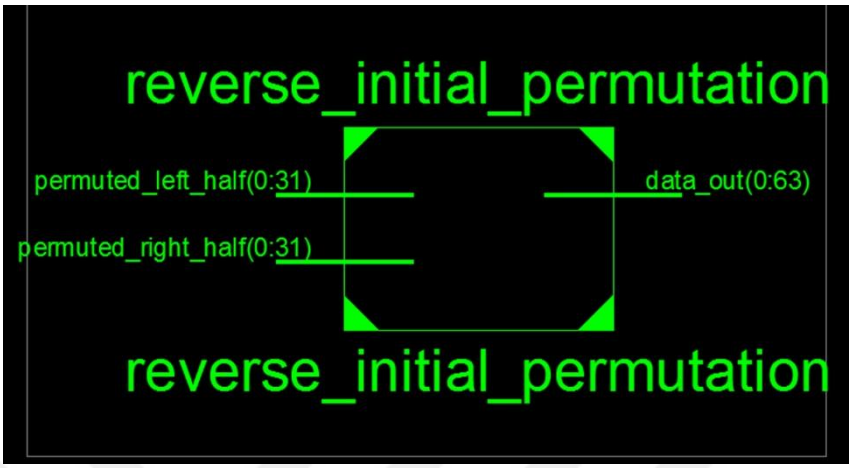

Figure 4.20. RTL Block of Final Permutation in DES Algorithm

Figure 4.21. below shows the Simulation results of DES Encryption Algorithm. For simulation of DES encryption algorithm, 64-bit input key is assigned as "ABCDEF1123456789" in hexadecimal, and the plain text input is assigned as the ASCII codes of "HARRANUN", “48415252414E554E” in hexadecimal. The output of DES encryption is generated as "ADC1358C846CE62D" in hexadecimal, which is called as cipher text. The resource use of the design is presented in Table 4.4.

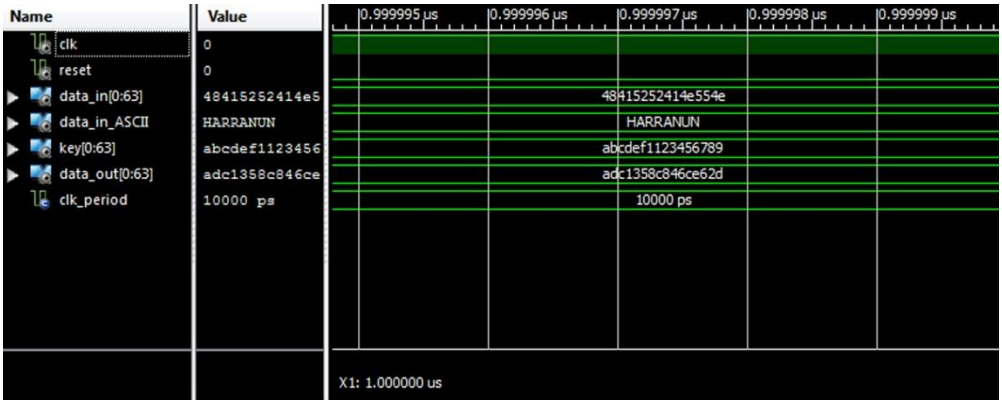

Figure 4.21. Simulation Results of DES Encryption Algorithm

Table 4.4. The resource use of DES Encryption Algorithm

DES Encryption algorithm	***Simulation Results***
Elaboration Time	0.670804 s
Total Signals	1010
Total Nets	7549
Total Signal Drivers	3373
Total Blocks	344
Total Primitive Blocks	246
Total Processes	3373
Total Traceable Variables	174
Total Scalar Nets and Variables	16273
Current Memory Usage	1026.8 MB

Figure 4.22. below shows the simulation results of DES decryption Algorithm, DES Decryption Algorithm block is simulated with the same 64-bit key used in the Encryption block and 64-bit cipher text input generated in the Encryption block which is "ADC1358C846CE62D" in hexadecimal. The plain text output is produced as "48415252414E554E" in hexadecimal. The resource use of the design is shown in Table 4.5.

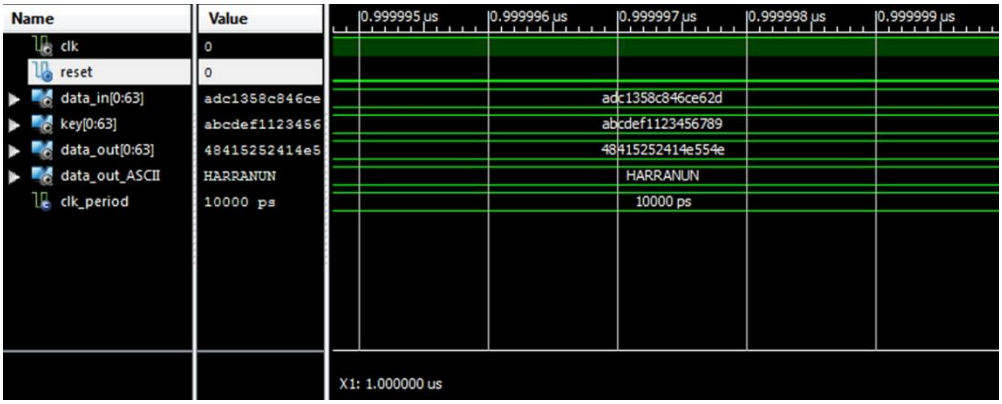

Figure 4.22. Simulation Results of DES Decryption Algorithm

Table 4.5. The resource use of DES Decryption Algorithm

DES Decryption algorithm	***Simulation Results***
Elaboration Time	0.655204 s
Total Signals	1014
Total Nets	7571
Total Signal Drivers	3429
Total Blocks	346
Total Primitive Blocks	248
Total Processes	3429
Total Traceable Variables	174
Total Scalar Nets and Variables	16295
Current Memory Usage	712.229 MB

Table 4.6 shows the complete summary of our implemented design, which describes the resource use for Encryption and Decryption of the DES Algorithm.

Table 4.6. The resource use of DES Encryption and Decryption Algorithms

Basis For Comparison in DES	***Encryption Result DES***	***Decryption Result DES***
Elaboration Time	0.670804 s	0.655204 s
Total Signals	1010	1014
Total Nets	7549	7571
Total Signal Drivers	3373	3429
Total Blocks	344	346
Total Primitive Blocks	246	248
Total Processes	3373	3429
Total Traceable Variables	174	174
Total Scalar Nets and Variables	16273	16295
Current Memory Usage	1026.8 MB	712.229 MB

The DES simulation result shows different time and memory usage of DES algorithms. DES requires 0.670804 seconds and Memory Usage of 1026.8 MB for encryption processes, and 0.655204 seconds and Memory Usage of 712.229 MB for decryption.

4.1.3. Software Comparison of AES and DES Algorithms

Both AES and DES algorithms were used as encryption and decryption. For testing both algorithms in the Xilinx ISE 14.7. Software with the target of XC6SLX45-3CSG324, ISIM Simulator was used. Software comparison of AES and DES Encryption and Decryption algorithms is presented in Table 4.7.

Table 4.7. Software Comparison of AES and DES Encryption and Decryption Algorithms

Basis For Comparison	***Software Comparison of AES and DES Algorithms***	
	AES Algorithm	***DES Algorithm***
Elaboration Time	1.248008 s	1.326008 s
Total Signals	128	2024
Total Nets	4023	15120
Total Signal Drivers	120	6802
Total Blocks	76	690
Total Primitive Blocks	38	494
Total Processes	117	6802
Total Traceable Variables	50	348
Total Scalar Nets and Variables	4777	32568
Current Memory Usage	1705.652 MB	1739.029 MB

As seen in Table 4.7., AES algorithm requires 1.248008 seconds and Memory Usage of 1705.652 MB and DES algorithm requires 1.326008 seconds and Memory Usage of 1739.029 MB. AES algorithm takes less time and less memory for both encryption and decryption processes compared to DES algorithm. It can be observed from the simulation results that AES is more robust. The encryption and decryption of AES algorithm are more secure.

4.2. Hardware Implementation

4.2.1 Hardware Implementation of AES

Since the Rijndael algorithm was confirmed as an Advanced Encryption Standard, the AES algorithm has played a significant role in the field of information protection. Implementation of the AES algorithm on FPGA has the advantages of fast, scalable, shorter production period, etc. Both AES and DES algorithms are implemented on FPGA.

The architecture supports varieties of AES encryption and decryption algorithm. VHDL code is programmed on the Xilinx ISE 14.7. software. Table 4.8. below shows the hardware resource use of the AES Algorithm for Encryption and Decryption. Test findings reveal that total signals are 128, total nets are 4023, total blocks are 76, Memory Usage is 34740 KB and the simulation time is 1.60681 seconds for AES Encryption and Decryption algorithms.

Table 4.8. Hardware Resource Use of AES Encryption and Decryption Algorithms

Basis For Comparison AES	***Encryption/decryption of AES Algorithms***
Memory Usage	34740 KB
Total Signals	128
Total Nets	4023
Total Blocks	76
Total Processes	117
Total Simulation Time	3 µs
Simulation Resource Usage	1.60681 s, 989630 KB

4.2.2 Hardware Implementation of DES

Security measures on the internet are provided considerable importance nowadays. The DES is one of the most popular forms of block encryption/decryption commonly used. This study implemented DES encryption and decryption algorithm with high-performance reconfigurable hardware.

Using Spartan-6 (XC6SLX45) family FPGAs, the proposed design is implemented and is among the quickest hardware implementations with much greater reliability as shown in Table 4.9.

Table 4.9. Hardware Resource Use of DES Encryption and Decryption Algorithms

Basis For Comparison DES	***Encryption and Decryption of DES Algorithms***
Memory Usage	38388 KB
Total Signals	2024
Total Nets	15120
Total Blocks	690
Total Processes	6802
Total Simulation Time	3 µs
Simulation Resource Usage	2.43362 s, 1022844 KB

Test findings reveal that total signals are 2024, total nets are 15120, total blocks are 690, Memory Usage is 38388 KB and the simulation time is 3 µ seconds for DES Encryption and Decryption algorithms.

4.2.3 Hardware Comparison of AES and DES Algorithms

The results of a hardware implementation of AES and DES Algorithms on the FPGA board of Spartan, show that Current Memory Usage is 34740 KB, the implementation time is 1.60681 second for both encryption/decryption processes of AES algorithm, while Current Memory Usage is 38388 KB and the implementation time is 2.43362 second for both encryption/decryption processes of the DES algorithm. The comparison of the results is shown in Table 4.10.

Table 4.10. Hardware Comparison of AES and DES Algorithms

Basis for Comparison	***Hardware Comparison of AES and DES Algorithm***	
	AES Algorithm	***DES Algorithm***
Memory Usage	34740 KB	38388 KB
Total Signals	128	2024
Total Nets	4023	15120
Total Blocks	86	690
Total Processes	117	6802
Total Simulation Time	3 µs	3 µs
Simulation Resource Usage	1.60681 s, 989630 KB	2.43362 s, 1022844 KB

These results suggest that, relative to the AES algorithm, the DES Algorithm takes more time for implementation. In comparison, the DES algorithm uses more memory space than the AES algorithm. It is inferred that AES is faster than the DES algorithm, and AES runs at a higher degree of protection relative to DES from a safety viewpoint.

5. CONCLUSION and SUGGESTIONS

5.1. Conclusion

The Encryption algorithm is an essential part of data security. Each cryptographic algorithm has weak and strong points. The cryptographic algorithm is chosen based on the specifications of the program or the application which would be used. This thesis analyzes the application of crypto techniques of both AES and DES algorithms on FPGA in terms of memory usage, hardware usage, and speed. Both AES and DES Algorithms are tested for encryption and decryption processing capacity. The speed and processing time of AES indicates that it is faster than DES. As expected, the design for AES uses less blocks, less signals and less net, while the use of memory is similar to the design for DES.

AES algorithm is more secure compared to DES algorithm. While AES has a choice of 128-bit, 192-bit or 256-bit key, DES has only 56-bit key. In terms of architecture, DES uses the division blocks before going through the encryption steps. However, AES uses series of permutation-substitution blocks to encrypt the plain text.

5.2. Suggestions

Implementing both cryptographic algorithms on a software and hardware framework that is suitable for communication technologies would provide the algorithms with strong support. In addition, advent in FPGA technologies would introduce new architectures and new symmetric encryption schemes.

REFERENCES

ABDULWAHID, M. N, ALI, A., ESPARHAM, B. and MARWAN, M., 2018. A Comparison of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish for Guessing Attacks Prevention. Journal of Computer Science Applications and Information Technology, 3(2): 1-7.

AGRAWAL, M. and MISHRA, P., 2012. A Comparative Survey on Symmetric Key Encryption Techniques. International Journal on Computer Science and Engineering (IJCSE), 4(5): 877-882.

AHMAD, N., HASAN, R. and JUBADI, W. M., 2010. Design of AES S-Box using combinational logic optimization. IEEE Symposium on Industrial Electronics and Applications (ISIEA), 3-5 Oct, Penang, p. 696-699.

ALAMMD, I. and KHAN, M., 2013. Performance and Efficiency Analysis of Different Block Cipher Algorithms of Symmetric Key Cryptography. International Journal of Advanced Research in Computer Science and Software Engineering, 3(10): 713-720.

ALIOTO, M. and ROCCHI, S., 2010. Differential Power Analysis Attacks to Precharged Buses: A General Analysis for Symmetric-Key Cryptographic Algorithms. IEEE Transactions on Dependable and Secure Computing, 7(3): 226-239.

BHANOT, R. and HANS, R., 2015. A Review and Comparative Analysis of Various Encryption Algorithms. International Journal of Security and Its Applications (IJSIA), 9(4): 289-306.

BHATELE, K., SINHAL, A. and PATHAK, M., 2012. A Novel Approach to the Design of a New Hybrid Security Protocol Architecture. IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT), 23-25 Aug, Ramanathapuram, p. 429-433.

BRINDHA, K., SHARMA, R. and SAINI, S., 2014. Use of Symmetric Algorithm for Image Encryption. International Journal of Innovative Research in Computer and Communication Engineering, 2(5): 4401-4407.

BUONO, D. and MENCAGLI, G., 2014. Run-time mechanisms for fine-grained parallelism on network processors: International Conference on High Performance Computing & Simulation (HPCS), 21-25 July, Bologna, p. 55-64.

GULERIA, S. and VATTA, S., 2013. TO Enhance Multimedia Security in Cloud Computing Environment using Crossbreed Algorithm. International Journal of Application or Innovation in Engineering and Management (IJAIEM), 2(6): 562-568.

GHODKE, M. B. and MALI, N. S., 2016. Implementation of Advanced Encryption Standard Algorithm for Communication Security Using FPGA. International Research Journal of Engineering and Technology (IRJET), 3(7): 1176-1179.

JEEVA, A., PALANISAMY, V. and KANAGARAM, K., 2012. Comparative Analysis of Performance Efficiency and Security Measures of Some Encryption Algorithms. International Journal of Engineering Research and Applications (IJERA), 2(3): 3033-3037.

KIM, C. H., 2012. Improved Differential Fault Analysis on AES Key Schedule. IEEE Transactions on Information Forensics and Security, 7(1): 41-50.

MAHAJAN, P. and SACHDEVA, A., 2013. A Study of Encryption Algorithms AES, DES and RSA for Security. Global Journal of Computer Science and Technology Network Web & Security, 13(15): 15-22.

MANDAL, P. C., 2012. Evaluation of performance of the Symmetric Key Algorithms: DES, 3DES, AES and Blowfish. Journal of Global Research in Computer Science, 3(8): 67-70.

PADMAPRIYA, A. and SUBHASRI, P., 2013. Cloud Computing: Security Challenges & Encryption Practices. International Journal of Advanced Research in Computer Science and Software Engineering, 3(3): 255-259.

PRASHANTI, G., DEEPTHI, S. and SANDHYA, R., 2013. A Novel Approach for Data Encryption Standard Algorithm. International Journal of Engineering and Advanced Technology (IJEAT), 2(5): 264-267.

PRITAMKUMAR, N. and VRUSHALI, G., 2015. Implementation of AES Algorithm on FPGA for Low Area Consumption. International Conference on Pervasive Computing (ICPC), 8-10 Jan, Pune, p. 1-4.

RIMAN, C. and ABI-CHAR, E., 2015. Comparative Analysis of Block Cipher-Based Encryption Algorithms: A Survey. Information Security and Computer Fraud, 3(1): 1-7.

SAKIYAMA, K., LI, Y. and OHTA, K., 2012. New Fault Based Side Channel Attack Using Fault Sensitivity. IEEE Transactions on Information Forensics and Security, 7(1): 88-97.

SETH, S. M. and MISHRA, R., 2011. Comparative Analysis of Encryption Algorithm For Data Communication. International Journal of Computer Science and Technology, 2(2): 292-294.

SETYANINGSIH, Y. and WARDOYO, R., 2017. Review of Image Compression and Encryption Techniques. International Journal of Advanced Computer Science and Applications, 8(2): 83-94.

SINDHU, S. and SINDHU, D., 2017. Cryptographic Algorithms: Applications in Network Security. International Journal of New Innovations in Engineering and Technology, 7(1): 18-28.

SINGH, G. and SUPRIY, A., 2013. A Study of Encryption Algorithms RSA, DES, 3DES and AES for Information Security. International Journal of Computer Applications, 67(19): 33-38.

SINGHAL, N. and RAINA, J., 2011. Comparative Analysis of AES and RC4 Algorithms for Better Utilization. International Journal of Computer Trends and Technology, 2(6): 177-181.

SONI, S., AGRAWAL, H. and SHARMA, M., 2012. Analysis and Comparison between AES and DES Cryptographic Algorithm. International Journal of Engineering and Innovative Technology (IJEIT), 2(6): 362-365.

SRINIVAS, B. L. and SHANBHAG, A., 2014. A Comparative Performance Analysis of DES and BLOWFISH Symmetric Algorithm. International Journal of Innovative Research in Computer and Communication Engineering, 2(5): 77-88.

YANG, H., OSTERWEIL, E., MASSEY, D. and MEMBER, S., 2011. Deploying Cryptography in Internet-Scale Systems: A Case Study on DNSSEC. IEEE Transactions on Dependable and Secure Computing, 8(5): 656-669.

CURRICULUM VITAE

PERSONAL INFORMATION

Name	:	Abdulsamad Ibrahim Hussein KURD
Nationality	:	Iraqi
Place of Birth and Date	:	Erbil. 20-5-1986
Phone Number	:	009647504603960
E-Mail	:	Resa4it@gmail.com

EDUCATION

Degree	**School/University**	**Year**
High School :	Cevahiri High School	2005
BSc. Degree :	University of Sulaimani	2009
MSc. Degree:	Harran University	2020

RESEARCH INTERESTS

2009 to now: Working in IT Department for Hawler Medical University.

FOREIGN LANGUAGE

English, Turkish, Arabic

PUBLICATIONS

KURD, A. and BEŞLİ, N. 2020. Analysis of the Cryptography Methods for Design of Crypto-Processor. International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), June 26-27, Ankara, Turkey, pp. 1-7.

APPENDIX

APPENDIX A: VHDL Code for AES

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use ieee.std_logic_unsigned.all;

ENTITY aes IS
PORT(
   clk              :  IN     std_logic;
   reset  :  IN     std_logic;
    encrypt_in128  : IN std_logic;
    encrypt_out128     : OUT std_logic;
    decrypt_in128  : IN std_logic;
    decrypt_out128     : OUT std_logic;
    busy        : OUT std_logic
   );
END aes;
ARCHITECTURE beh OF aes IS
COMPONENT round
PORT(
 clk              :  IN     std_logic;
   reset  :  IN     std_logic;
   d_in   :  IN  std_logic_vector(127 downto 0);
   key              :  IN  std_logic_vector(127 downto 0);
   last_mux_sel:  IN  std_logic;
   data_out         :  OUT std_logic_vector(127 downto 0)
   );
END COMPONENT;
COMPONENT inv_round
PORT(
 clk              :  IN     std_logic;
   reset  :  IN     std_logic;
```

```
    d_in   :   IN  std_logic_vector(127 downto 0);
    key              :   IN  std_logic_vector(127 downto 0);
    last_mux_sel:   IN  std_logic;
    data_out         :   OUT std_logic_vector(127 downto 0)
    );
END COMPONENT;

SIGNAL key_sel         : std_logic;
--SIGNAL round0_out       : std_logic_vector(127 downto 0);
SIGNAL Encryption_aes_Out       : std_logic_vector(127 downto 0);
SIGNAL Decryption_aes_Out       : std_logic_vector(127 downto 0);
SIGNAL key_in_128             : std_logic_vector(127 downto 0);
SIGNAL Encryption_aes     : std_logic_vector(127 downto 0);
SIGNAL key             : std_logic_vector(127 downto 0);
SIGNAL Decryption_aes     : std_logic_vector(127 downto 0);
--SIGNAL round1_10_out    : std_logic_vector(127 downto 0);
--Encrypt
SIGNAL Round10_Encryption_aes_Out    : std_logic_vector(127 downto 0);
SIGNAL Round10_Decryption_aes_Out    : std_logic_vector(127 downto 0);
SIGNAL round_constant   : std_logic_vector(7 downto 0);
SIGNAL data_sel        : std_logic_vector(1 downto 0);
SIGNAL load_data       : std_logic;
SIGNAL load_key        : std_logic;
SIGNAL last_mux_sel     : std_logic;
SIGNAL encrypt_data_tx  : std_logic_vector(127 downto 0);
SIGNAL decrypt_data_tx  : std_logic_vector(127 downto 0);
SIGNAL key_reg       : std_logic_vector(127 downto 0);
SIGNAL Key_Log           : std_logic;
SIGNAL Key_sig      : std_logic;
SIGNAL encrypt_sig        : std_logic;
SIGNAL decrypt_sig      : std_logic;
SIGNAL enorde : std_logic;
BEGIN
Prescaler: process(reset,clk)
          begin
```

```
                    if rising_edge(clk) then
                     if reset = '1' then
                            if Encryption_aes < "10111110101111000001000000" then
                                    Encryption_aes <= Encryption_aes + 1;
                            else
                                    encrypt_sig <= not encrypt_sig;
                                    Encryption_aes <= (others => '0');
                            end if;
                            end if;
                            if reset = '1' then
                            if Decryption_aes < "01010010111111010000001110" then
                                    Decryption_aes <= Decryption_aes + 1;
                            else
                                    decrypt_sig <= not decrypt_sig;
                                    Decryption_aes <= (others => '0');
                            end if;
                    end if;
if reset = '1' then
                            if key_in_128 < "01010010111111010000001110" then
                                    --key_in_64 <=  + 1;
                            else
                                    --Key_sig <= not Key_sig;
                            --      key_in_64 <= (others => '0');
                            end if;
                    end if;
                    end if;
            end process Prescaler;
            encrypt_out128 <= encrypt_sig;
            decrypt_out128 <= decrypt_sig;

Encrypt: round
 PORT MAP(
            clk         =>  clk,
            reset       =>  reset,
            d_in        =>  Encryption_aes,
```

```
        key          =>  key,
        last_mux_sel=>  last_mux_sel,
        data_out    =>  Round10_Encryption_aes_Out
        );

Decrypt:inv_round
 PORT MAP(
        clk          =>  clk,
        reset        =>  reset,
        d_in         =>  Decryption_aes,
        key          =>  key,
        last_mux_sel=>  last_mux_sel,
        data_out    =>  Round10_Decryption_aes_Out
        );

key_generator: key_schedule
PORT MAP(
        clk                =>  clk,
        reset              =>  reset,
        key_in             =>  key_reg,
        key_out            =>  key,
        key_sel        =>  key_sel,
    enorde         =>  enorde,
        round_constant    =>  round_constant,
        load_key           =>  load_key
        );

Encryption_aes_Out <= Encryption_aes XOR key;
Decryption_aes_Out <= Decryption_aes XOR key;
encrypt_data_tx<= Encryption_aes;
decrypt_data_tx<= Decryption_aes;
END beh;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

```
USE work.ALL;

ENTITY key_schedule IS
PORT(
   clk                  :  IN  std_logic;
   reset      :  IN  std_logic;
   key_in               :  IN  std_logic_vector(127 downto 0);
   key_out              :  OUT std_logic_vector(127 downto 0);
   key_sel        :  IN  std_logic;
   enorde         :  IN  std_logic;
   round_constant :          IN  std_logic_vector(7 downto 0);
   load_key   :    IN  std_logic
   );
END key_schedule;
ARCHITECTURE beh OF key_schedule IS
COMPONENT s_box_4
PORT(
   s_box_4_in    : IN  std_logic_vector(31 downto 0);
   s_box_4_out   :OUT std_logic_vector(31 downto 0)
   );
END COMPONENT;

SIGNAL U         : std_logic_vector(31 downto 0);
SIGNAL left_shift : std_logic_vector(31 downto 0);
SIGNAL sbox       : std_logic_vector(31 downto 0);
SIGNAL key_reg_in  : std_logic_vector(127 downto 0);
SIGNAL next_key    : std_logic_vector(127 downto 0);
SIGNAL key_reg_out : std_logic_vector(127 downto 0);
SIGNAL upperbyte   : std_logic_vector(7 downto 0);

TYPE word_array is ARRAY (3 downto 0) OF std_logic_vector(31 downto 0);
SIGNAL key_word, next_key_word : word_array;
BEGIN
key_reg_in <= key_in  WHEN key_sel='0' ELSE -- when 1st round
                next_key;       --for all other round rounds

key_0:PROCESS(reset, clk)
BEGIN
   IF(reset='1') THEN
```

```
            key_reg_out <= (others =>'0');
    ELSIF(clk'event AND clk='1') THEN
            IF(load_key='1') THEN
               key_reg_out <= key_reg_in;
            END IF;
    END IF;
END PROCESS key_0;
--mapping a vector into array of words
key_word(0) <= key_reg_out(127 downto 96);
key_word(1) <= key_reg_out(95 downto 64);
key_word(2) <= key_reg_out(63 downto 32);
          key_word(3) <= key_reg_out(31 downto 0);
 next_key_word(3) <= key_word(3) XOR key_word(2) WHEN enorde='1' ELSE
                key_word(3) XOR key_word(2) XOR key_word(1) XOR key_word(0)
 XOR U;
 next_key_word(2) <= key_word(2) XOR key_word(1) WHEN enorde='1' ELSE
                  key_word(2) XOR key_word(1) XOR key_word(0) XOR U;
 next_key_word(1) <= key_word(1) XOR key_word(0) WHEN enorde='1' ELSE
                   key_word(1) XOR key_word(0) XOR U;
 next_key_word(0) <= key_word(0) XOR U WHEN enorde='1' ELSE
                   key_word(0) XOR U;


 next_key <= next_key_word(0) & next_key_word(1) & next_key_word(2) & next_key_word(3);

 -- calculation of U

 left_shift <= (next_key_word(3)(23 downto 16) &
             next_key_word(3)(15 downto 8) &
             next_key_word(3)(7 downto 0) &
             next_key_word(3)(31 downto 24)) WHEN enorde='1' ELSE
             (key_word(3)(23 downto 16) &
             key_word(3)(15 downto 8) &
             key_word(3)(7 downto 0) &
             key_word(3)(31 downto 24));

 --key Sub Byte transformation

 sbox_q: s_box_4
```

```
PORT MAP(
      s_box_4_in   => left_shift,
      s_box_4_out  => sbox
      );

--XOR the upperbyte and round constant
upperbyte <= sbox(31 downto 24) XOR round_constant;
U <= upperbyte & sbox(23 downto 0);
key_out <= key_reg_out;
END beh;

      LIBRARY ieee;
      USE ieee.std_logic_1164.ALL;
      USE work.ALL;

      ENTITY key_schedule IS
      PORT(
        clk                 :  IN  std_logic;
        reset      :  IN  std_logic;
        key_in              :  IN  std_logic_vector(127 downto 0);
        key_out             :  OUT std_logic_vector(127 downto 0);
        key_sel        :  IN  std_logic;
        enorde         :  IN  std_logic;
        round_constant :          IN  std_logic_vector(7 downto 0);
        load_key    :     IN  std_logic
        );
      END key_schedule;

      ARCHITECTURE beh OF key_schedule IS

      COMPONENT s_box_4
      PORT(
        s_box_4_in    : IN  std_logic_vector(31 downto 0);
        s_box_4_out   :OUT std_logic_vector(31 downto 0)
        );
      END COMPONENT;

      SIGNAL U         : std_logic_vector(31 downto 0);
      SIGNAL left_shift : std_logic_vector(31 downto 0);
```

```
  SIGNAL sbox       : std_logic_vector(31 downto 0);

  SIGNAL key_reg_in  : std_logic_vector(127 downto 0);
  SIGNAL next_key    : std_logic_vector(127 downto 0);
  SIGNAL key_reg_out : std_logic_vector(127 downto 0);
  SIGNAL upperbyte   : std_logic_vector(7 downto 0);

  TYPE word_array is ARRAY (3 downto 0) OF std_logic_vector(31 downto 0);
  SIGNAL key_word, next_key_word : word_array;

  BEGIN

  key_reg_in <= key_in  WHEN key_sel='0' ELSE -- when 1st round
                next_key;       --for all other round rounds

  key_0:PROCESS(reset, clk)
  BEGIN
     IF(reset='1') THEN
           key_reg_out <= (others =>'0');
     ELSIF(clk'event AND clk='1') THEN
           IF(load_key='1') THEN
              key_reg_out <= key_reg_in;
           END IF;
     END IF;
  END PROCESS key_0;

  key_word(0) <= key_reg_out(127 downto 96);
  key_word(1) <= key_reg_out(95 downto 64);
  key_word(2) <= key_reg_out(63 downto 32);
  key_word(3) <= key_reg_out(31 downto 0);
next_key_word(3);
   next_key_word(3) <= key_word(3) XOR key_word(2) WHEN enorde='1' ELSE
                       key_word(3) XOR key_word(2) XOR key_word(1) XOR key_word(0)
   XOR U;
   next_key_word(2) <= key_word(2) XOR key_word(1) WHEN enorde='1' ELSE
                        key_word(2) XOR key_word(1) XOR key_word(0) XOR U;
   next_key_word(1) <= key_word(1) XOR key_word(0) WHEN enorde='1' ELSE
                        key_word(1) XOR key_word(0) XOR U;
   next_key_word(0) <= key_word(0) XOR U WHEN enorde='1' ELSE
```

```
                    key_word(0) XOR U;
        next_key <= next_key_word(0) & next_key_word(1) & next_key_word(2) &
        next_key_word(3);

        left_shift <= (next_key_word(3)(23 downto 16) &
                next_key_word(3)(15 downto 8) &
                next_key_word(3)(7 downto 0) &
                next_key_word(3)(31 downto 24)) WHEN enorde='1' ELSE
                (key_word(3)(23 downto 16) &
                key_word(3)(15 downto 8) &
                key_word(3)(7 downto 0) &
                key_word(3)(31 downto 24));

        --key subbyte transformation

        sbox_q: s_box_4
        PORT MAP(
            s_box_4_in   => left_shift,
            s_box_4_out  => sbox
            );
        key_out <= key_reg_out;
END beh;
```

APPENDIX B: VHDL Code For DES

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use IEEE.std_logic_arith.ALL;
use ieee.std_logic_unsigned.all;
ENTITY DES IS
PORT(
  clk              :  IN     std_logic;
   reset  :  IN     std_logic;
    encrypt_in64 : IN std_logic;
```

```
    encrypt_out64     : OUT std_logic;
    decrypt_in64  : IN std_logic;
    decrypt_out64    : OUT std_logic;
                key_in64  : IN std_logic;
    busy        : OUT std_logic
   );
END DES;

ARCHITECTURE beh OF DES IS
COMPONENT encrypt
PORT(
   data_in: in std_logic_vector(0 to 63);
          key: in std_logic_vector(0 to 63);
          clk                    :   IN      std_logic;
   reset   :   IN      std_logic;
           busy        : OUT std_logic;
          data_out: out std_logic_vector(0 to 63)
   );
END COMPONENT;

COMPONENT decrypt
PORT(
   data_in: in std_logic_vector(0 to 63);
          key: in std_logic_vector(0 to 63);
          clk                    :   IN      std_logic;
   reset   :   IN      std_logic;
           busy        : OUT std_logic;
          data_out: out std_logic_vector(0 to 63)
   );
END COMPONENT;
SIGNAL Data_Out_Encrypt       : std_logic_vector(63 downto 0);
SIGNAL Data_Out_Decrypt       : std_logic_vector(63 downto 0);
SIGNAL Data_in_to_Encrypt     : std_logic_vector(63 downto 0);
SIGNAL key              : std_logic_vector(63 downto 0);
SIGNAL key_in_64              : std_logic_vector(63 downto 0);
SIGNAL Data_in_to_Decrypt     : std_logic_vector(63 downto 0);
SIGNAL Key_sig      : std_logic;
SIGNAL encrypt_sig         : std_logic;
SIGNAL decrypt_sig      : std_logic;
```

```
BEGIN

Prescaler: process(reset,clk)
        begin
                if rising_edge(clk) then
                 if reset = '1' then
                        if Data_in_to_Encrypt < "10111110101111000001000000" then
                                Data_in_to_Encrypt <= Data_in_to_Encrypt + 1;
                        else
                                encrypt_sig <= not encrypt_sig;
                                Data_in_to_Encrypt <= (others => '0');
                        end if;
                        end if;
                        if reset = '1' then
                        if Data_in_to_Decrypt < "01010010111110100000001110" then
                                Data_in_to_Decrypt <= Data_in_to_Decrypt + 1;
                        else
                                decrypt_sig <= not decrypt_sig;
                                Data_in_to_Decrypt <= (others => '0');
                        end if;
                end if;
        if reset = '1' then
                        if key_in_64 < "01010010111110100000001110" then
                                --key_in_64 <=  + 1;
                        else
                                --Key_sig <= not Key_sig;
                        --      key_in_64 <= (others => '0');
                        end if;
                end if;
                end if;

        end process Prescaler;
        encrypt_out64 <= encrypt_sig;
        decrypt_out64 <= decrypt_sig;
        --key_in_64 <= Key_sig;

Encrypt_DES: encrypt
 PORT MAP(
```

```
            data_in      =>  Data_in_to_Encrypt,
            key          =>  key,
            clk                  =>  clk,
    reset    => reset,
             busy       => busy,
            data_out    =>  Data_Out_Encrypt
            );
 Decrypt_DES:decrypt
 PORT MAP(
            data_in      =>  Data_in_to_Decrypt,
            key          =>  key,
            clk                  =>  clk,
    reset    => reset,
             busy       => busy,
            data_out    =>  Data_Out_Decrypt
            );
END beh;
library ieee;
use ieee.std_logic_1164.all;
entity round is
port(       left_plain: in std_logic_vector(0 to 31);
            right_plain: in std_logic_vector(0 to 31);
            subkey: in std_logic_vector(0 to 47);
            left_data_out: out std_logic_vector(0 to 31);
            right_data_out: out std_logic_vector(0 to 31));
end round;
architecture behavior of round is
            component f
            port(       data_in: in std_logic_vector(0 to 31);
                        key: in std_logic_vector(0 to 47);
                        data_out: out std_logic_vector(0 to 31));
            end component;
            component xor_32_bits
            port(       data_in: in std_logic_vector(0 to 31);
                        key: in std_logic_vector(0 to 31);
                        data_out: out std_logic_vector(0 to 31));
            end component;
            signal after_f: std_logic_vector(0 to 31);
begin
```

```
        s1: f port map(
                data_in=>right_plain,
                key=>subkey,
                data_out=>after_f);
        s2: xor_32_bits port map(
                data_in=>after_f,
                key=>left_plain,
                data_out=>right_data_out);
        left_data_out<=right_plain;
end;
library ieee;
use ieee.std_logic_1164.all;
entity subkey_production is
generic(
        shifting_parameter: in std_logic_vector(0 to 1);
        left_or_right: in std_logic_vector(0 to 0)); --0 represents left shift while 1 right shift
port(   left_key_in: in std_logic_vector(0 to 27);
        right_key_in: in std_logic_vector(0 to 27);
        subkey: out std_logic_vector(0 to 47);
        left_key_out: out std_logic_vector(0 to 27);
        right_key_out: out std_logic_vector(0 to 27));
end subkey_production;
        component key_permutation_2 is
        port(   left_half: in std_logic_vector(0 to 27);
                right_half: in std_logic_vector(0 to 27);
                permuted_key: out std_logic_vector(0 to 47));
        end component;
        signal left_half_shifted: std_logic_vector(0 to 27);
        signal right_half_shifted: std_logic_vector(0 to 27);
        signal a: std_logic_vector(0 to 1);

begin
        shift_by_1: if (shifting_parameter = "01") generate
                left_shifting: if (left_or_right = "0") generate
                        s11: left_shift_by_1 port map(
                                data_in=> left_key_in,
                                data_out=> left_half_shifted);

                        s12: left_shift_by_1 port map(
```

```
                    data_in=> right_key_in,
                    data_out=> right_half_shifted);
        end generate left_shifting;
        right_shifting: if (left_or_right = "1") generate
            s13: right_shift_by_1 port map(
                    data_in=> left_key_in,
                    data_out=> left_half_shifted);

            s14: right_shift_by_1 port map(
                    data_in=> right_key_in,
                    data_out=> right_half_shifted);
        end generate right_shifting;
    end generate shift_by_1;
    shift_by_2: if (shifting_parameter = "10") generate
        left_shifting: if (left_or_right = "0") generate
            s21: left_shift_by_2 port map(
                    data_in=> left_key_in,
                    data_out=> lefT_half_shifted);
            s22: left_shift_by_2 port map(
                    data_in=> right_key_in,
                    data_out=> right_half_shifted);
        end generate left_shifting;
        right_shifting: if (left_or_right = "1") generate
            s23: right_shift_by_2 port map(
                    data_in=> left_key_in,
                    data_out=> left_half_shifted);
            s24: right_shift_by_2 port map(
                    data_in=> right_key_in,
                    data_out=> right_half_shifted);
        end generate right_shifting;
    end generate shift_by_2;
    left_key_out<= left_half_shifted;
    right_key_out<= right_half_shifted;
end;
```