



T.C

ALTINBAŞI UNIVERSITY

Graduate School of Sciences and Engineering

Electrical and Computer Engineering

**DESIGN, IMPLEMENTATION AND PERFORMANCE
ANALYSIS OF AES WITH ONE WAY ENCRYPTION
SYSTEM USING LABVIEW BASED FPGA**

SARA ABDULHALEM AL-SHAREA

Master Thesis

Thesis Supervisor:

Prof. Dr. Oguz BAYAT

Istanbul, 2019

**DESIGN, IMPLEMENTATION AND PERFORMANCE ANALYSIS OF
AES WITH ONE WAY ENCRYPTION SYSTEM USING LABVIEW
BASED FPGA**

By

SARA ABDULHALEEM AL-SHAREA

Electrical and Computer Engineering

Submitted to the Graduate School of Science and Engineering

In partial fulfillment of the requirements for the degree of

Master of Science

ALTINBAŞ UNIVERSITY

2019

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Oguz BAYAT

Supervisor

Examining Committee Members (first name belongs to the chairperson of the jury and the second name belongs to supervisor)

Prof. Dr. Oguz BAYAT

School of Engineering and
Natural Science,

Altinbas University

Asst. Prof. Dr. Muhammad ILYAS

School of Engineering and
Natural Science,

Altinbas University

Asst. Prof. Dr. Adil Deniz DURU

Faculty of Sport Sciences,

Marmara University

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Catatay AYDIN

Head of Department

Approval Date of Graduate School of
Science and Engineering: ____/____/____

Prof. Dr. Oguz BAYAT

Director

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

SARA ABDULHALEEM AL-SHAREA

DEDICATION

To my late father, dear mother, husband, children and my brothers, To my esteemed supervisor Dr. Oguz Bayat ,To my dear friends and all who have helped me to finish this research.



ACKNOWLEDGEMENTS

First of All, thanks to Almighty Allah who has given us courage, wisdom to complete this work and patience to face all difficulties.

Foremost, I would like to express my sincere gratitude to my advisor Asst. Prof. Dr. Oguz Bayat for the continuous support of my study and research, for his patience, motivation, enthusiasm and immense knowledge. This work would not have been possible without his guidance, support and encouragement. Under his guidance I successfully overcame many difficulties and learned a lot. His invaluable guidance and support at every stage has led to successful conclusion of the study.

I want to convey my special thanks to my mother, for her love and continuous encouragement for me, she is the most important person in my world and I dedicate this thesis to her. And I would like to thank my husband Amro, for supporting me and give me the strength all the time and to my lovely children Taleen and Taim, for being in my life and all my friends for Their call and moral support have been invaluable at every stage of my life. Thank you all for standing by me at all times.

ABSTRACT

DESIG, IMPLEMENTATION AND PERFORMANCE ANALYSIS OF AES WITH ONE WAY ENCRYPTION SYSTEM USING LABVIEW BASED FPGA

SARA ABDULHALEEM AL-SHAREA,

M.Sc., Electrical and Computer Engineering, Altınbaş University,

Supervisor: Prof. Dr. Oguz BAYAT

Date: October2019

Pages: 110

The meaning of security is to protect something important from not authorized peoples. Cryptography is one of the security systems. Many applications could be used in the security systems like military applications, email, banking, information technology of database. The proposed system uses advanced encryption standard with secure hash algorithms to produce a strong ciphertext. Moreover, this thesis make the combination of hash functions which is a one way encryption, as a key with Advanced Encryption Standard (AES) algorithm which is a two way encryption as a ciphering generator to obtain confidentiality and integrity. The proposed system combining between the advanced encryption standard with secure hash algorithms based on Labview software and hardware implementation using Xilinx FPGA card. System generator was made to connect the software of the proposed system with hardware platform based on MATLAB system generation and in Labview platform in MATLAB script. The simulation

results show the good ideas and powerful form of strength of security with different form and size of plain text with both SHA-1 and SHA-2 of size extend from (1-bits to 2256 bits) and from these results hardware implementation achieved by SPARTAN 3AN with clock of 50 MHz The proposed system was designed based on FPGA, simulated using MATLAB and Xilinx-ISE 14.7 and then implemented in Spartan-3A 700AN FPGA. The proposed hash algorithms in this thesis obtained with laptop specifications: CPU is 2.5 GHz, core i5, memory RAM 4G Bytes and system type 64 bits. The implementation completed with Labview version of 2013 32-bits, Xilinx ISE 14.7 and FPGA Spartan 3A 700AN. Well results obtaining and a satisfying security level of the proposed work with respect to security statistical test with pass percentage of 90%.

Keywords: Advanced Encryption Standard (AES), Cryptanalysis, Field programmable gate array (FPGA), LabView, Secure Hash Algorithm (SHA-1), Secure Hash Algorithm (SHA-2), and Security.

TABLE OF CONTENTS

| | <u>Pages</u> |
|--|--------------|
| ABSTRACT..... | vii |
| LIST OF TABLES | xiii |
| LIST OF FIGURES | xiii |
| LIST OF ABBREVIATIONS | xvii |
| LIST OF SYMBOLS | xix |
| 1. INTRODUCTION..... | 1 |
| 1.1 GENERAL INTRODUCTION | 1 |
| 1.2 LITERATURE SURVEY | 2 |
| 1.3 AIM OF WORK..... | 4 |
| 1.4 THESIS CONTENT | 5 |
| 2. AES AND HASH FUNCTION ALGORITHM..... | 6 |
| 2.1 INTRODUCTION..... | 6 |
| 2.2 CRYPTOGRAPHY..... | 6 |
| 2.2.1 Authentication | 8 |
| 2.2.2 Integrity | 9 |
| 2.2.3 Digital Signiture | 10 |
| 2.2.4 Confidentiality..... | 10 |
| 2.3 ADVANCED ENCRYPTION STANDARD (AES) | 11 |
| 2.3.1 AES Encryption | 13 |
| 2.3.2 AES Decryption | 18 |
| 2.4 BLOCK CIPHER MODES OF OPERATION..... | 20 |
| 2.4.1 ELECTRONIC CODEBOOK MODE (ECB) | 20 |
| 2.4.2 CIPHER BLOCK CHAINING (CBC) MODE..... | 21 |
| 2.4.3 CIPHER FEEDBACK MODE (CFB) | 22 |
| 2.4.4 OUTPUT FEEDBACK (OFB) MODE..... | 23 |

| | | |
|-----------|---|-----------|
| 2.4.5 | COUNTER (CTR) | 24 |
| 2.5 | SECURE HASH ALGORITHM (SHA)..... | 25 |
| 2.5.1 | Secure Hash Algorithm (SHA-1)..... | 25 |
| 2.5.2 | SHA-2 | 29 |
| 2.6 | ATTACKS ON THE PROPOSED SYSTEM | 30 |
| 2.7 | FIELD PROGRAMMABLE GATE ARRAY (FPGA) | 33 |
| 2.8 | ENCRYPTION QUALITY MEASURING TECHNIQUE | 35 |
| 3. | SOFTWARE IMPLEMENTATION OF AES AND HASH FUNCTION..... | 38 |
| 3.1 | INTRODUCTION..... | 38 |
| 3.2 | SOFTWARE IMPLEMENTATION..... | 38 |
| 3.2.1 | AES Implementation by LabView | 38 |
| 3.2.2 | AES Encryption process by LabView | 39 |
| 3.2.3 | AES Decryption process by LabView..... | 44 |
| 3.3 | IMPLEMENTATION OF HASH FUNCTIONS BY LABVIEW | 47 |
| 3.3.1 | Implementation of SHA-1 by LabView | 47 |
| 3.3.2 | Implementation of SHA-2 by LabView | 52 |
| 3.4 | COMPARATIVE ANALYSIS | 58 |
| 4. | SOFTWARE AND HARDWARE IMPLEMENTATION RESULT..... | 60 |
| 4.1 | INTRODUCTION..... | 60 |
| 4.2 | AES AND HASH FUNCTIONS SOFTWARE RESULT..... | 60 |
| 4.3 | IMPLEMENTATION OF THE PROPOSED ALGORITHM ON FPGA..... | 68 |
| 4.4 | MAPPING THE PROPOSED ALGORITHM ON FPGA..... | 70 |
| 4.5 | THE PROPOSED SYSTEM IMPLEMENTATION ON SPARTAN-3A 700AN PLATFOR | 73 |
| 4.6 | STATISTICAL TESTS..... | 75 |
| 5. | SOFTWARE AND HARDWARE IMPLEMENTATION RESULT..... | 78 |
| 5.1 | CONCLUSIONS | 78 |
| 5.2 | FUTURE WORK | 79 |

| | |
|------------------------------|-----------|
| REFERENCES..... | 80 |
| APPENDIX A | 84 |
| APPENDIX B | 85 |
| APPENDIX C | 88 |
| APPENDIX D | 94 |
| CURRICULUM VITAE..... | 96 |



LIST OF TABLES

| | <u>Pages</u> |
|--|---------------------|
| Table 2.1: Shift offsets for different block lengths | 15 |
| Table 2.2: Primitive logic functions used in SHA-1 | 28 |
| Table 2.3: The run length boundaries | 37 |
| Table 3.1: Comparative analysis of the proposed algorithm | 59 |
| Table 4.1: (a) Encryption and (b) Decryption Process Utilization | 71 |
| Table 4.2: FIPS PUB 140-1 statistical tests of the proposed system. | 76 |

LIST OF FIGURES

| | <u>Pages</u> |
|--|--------------|
| Figure 2.1: Encryption and decryption | 6 |
| Figure 2.2: Example on Authentication | 8 |
| Figure 2.3: Message integrity | 9 |
| Figure 2.4: Digital Signature Creation..... | 10 |
| Figure 2.5: Message Confidentiality achieved by encrypt/decrypt | 11 |
| Figure 2.6: General structure of AES algorithm..... | 12 |
| Figure 2.7: Sub-bytes transformation [5]..... | 14 |
| Figure 2.8: Shift Rows cyclically shift the last rows in the State | 15 |
| Figure 2.9: General Block Diagram of The Key Expansion [14]..... | 17 |
| Figure 2.10: Inverse Shift Rows transformation..... | 18 |
| Figure 2.11: ECB mode | 21 |
| Figure 2.12: CBC Mode..... | 22 |
| Figure 2.13: CFB Mode | 23 |
| Figure 2.14: Output Feedback (OFB) | 24 |
| Figure 2.15: Counter (CTR) Mode | 25 |
| Figure 2.16: Message digests construction [21] | 26 |

| | |
|---|----|
| Figure 2.17: SHA-1 Compression Function [21] | 29 |
| Figure 2.18: Main Structure of The FPGA [32] | 34 |
| Figure 3.1: Key Generation in LabView..... | 40 |
| Figure 3.2: AES Encryption Process by Labview | 43 |
| Figure 3.3: AES Decryption Process by Labview | 46 |
| Figure 3.4: Initial Vectors of SHA-1 | 47 |
| Figure3.5: SHA-1 core word calculation | 48 |
| Figure 3.6: SHA-1 calculations..... | 49 |
| Figure3.7: Iterations of the case structure..... | 50 |
| Figure 3.8: SHA-1 by Labview..... | 51 |
| Figure 3.9: Message Preparation..... | 52 |
| Figure3.10: Converting Array..... | 53 |
| Figure3.11: SHA-2 core Entry | 53 |
| Figure 3.12: Working Array | 54 |
| Figure 3.13: SHA-2 Calculations..... | 55 |
| Figure 3.14: Message Digest..... | 56 |
| Figure 3.15: SHA-2 by Labview..... | 57 |

| | |
|---|----|
| Figure 4.1: Testing AES and SHA-1 (capital letters) | 61 |
| Figure 4.2: Testing AES and SHA-1 (small letters) | 62 |
| Figure 4.3: Testing AES and SHA-1 (symbols and numbers) | 62 |
| Figure 4.4: Testing AES and SHA-1 (ECB mode) | 63 |
| Figure 4.5: Testing AES and SHA-1 (CFB mode) | 64 |
| Figure 4.6: Testing AES-256 and SHA-2 (capital letters) | 65 |
| Figure 4.7: Testing AES-256 and SHA-2 (small letters) | 65 |
| Figure 4.8: Testing AES-256 and SHA-2 (symbols and numbers) | 66 |
| Figure 4.9: Testing AES-256 and SHA-2 (ECB mode) | 67 |
| Figure 4.10: Testing AES-256 and SHA-2 (CFB mode) | 67 |
| Figure 4.11: Proposed System Black Box | 68 |
| Figure 4.12: Encryption Process of The Proposed System..... | 69 |
| Figure 4.13: Decryption process of The Proposed System..... | 69 |
| Figure 4.14:(a) Simulation Output of The Encryption and (b) Decryption Process | 70 |
| Figure 4.15: JTAG Cable | 73 |
| Figure 4.16: Hardware Co-simulation | 74 |
| Figure 4.17: The bits stream downloading to the hardware device through the JTAG cable..... | 74 |

Figure 4.18: Connection between the computer and Spartan-3AN board 75



LIST OF ABBREVIATIONS

| | |
|---------|---|
| AES | : Advanced Encryption Standard |
| ALU | : Arithmetic Logic Unit |
| ASCII | : American Standard Code for Information Interchange |
| ASIC | : Application Specification Related Circuit |
| CBC | : Cipher Block Chaining |
| CFB | : Cipher Feedback |
| CLBs | : Configurable Logic Blocks |
| CTR | : Counter |
| DES | : Data Encryption Standard |
| DSA | : Digital Signature Algorithm |
| ECB | : Electronic Code Book |
| FF | : Flip-Flop |
| FIPS | : Federal Information Processing Standard |
| FIPSPUB | : Federal Information Processing Standard Publication |
| FPGA | : Field Programmable Gate Array |
| GUI | : Graphical User Interface |
| Hwcosm | : Hardware Co-Simulator |
| ISE | : Integrated Synthesis Environment |
| IV | : Initial Vector |

JTAG : Joint Test Action Group

LabVIEW : Laboratory Virtual Instrument Engineering Workbench

LUT : Look Up Table

MAC : Message Authentication Code

Mod : Modulo

MUX : Multiplexer

NIST : National Institute of Standards and Technology

NSA : National Security Agency

P : Poker Test

PC : Personal Computer

RAM : Random Access Memory

RSA : Rivest, Shamir and Adleman

RTL : Register Translate Level

SHA : Secure Hash Algorithm

SHA-0 : Secure Hash Algorithm 0

SHA-1 : Secure Hash Algorithm 1

SHA-2 : Secure Hash Algorithm 2

SHA-3 : Secure Hash Algorithm 3

VHDL : Very high speed hardware description language

LIST OF SYMBOLS

| | |
|----------------------|---|
| * | : Addition, multiplication, or some other general mathematical operations |
| * | : Multiplication operation in mathematics |
| \oplus | : Xoring |
| \ll | : Shifting left |
| = | : Equality sign |
| \gg | : Shifting right |
| A,B,C,D,E | : Temporary vectors for SHA-1 |
| A,B,C,D,E,F,G,H | : Temporary vectors for SHA-2 256 |
| AND | : Login And |
| H | : Hash code |
| H (M), h_t | : Hash code of a message |
| H1,H2,H3,H4,H5 | : SHA-1 initial vectors |
| H1,H2,H3,H4,H5,H6,H7 | : Initial vectors of SHA-2 256 |
| I | : Iteration |
| i , j | : Used for looping |
| IV | : The random Initial Vector |
| Kt | : The constant vectors for hashing |
| L | : Length of message |

| | |
|--------------|--|
| M,M1,M2,M' | : Message |
| Maj | : Majority |
| Oj | : Plaintext message |
| P | : Probability of finding birthday attack |
| P,Q,R,S | : Primitive logic functions |
| ROT | : Rotation |
| S1 | : Register for hash calculations |
| SHF | : Shifting |
| T | : Iteration, round |
| S0 | : Register for hash calculations |
| TEMP | : Temporary register |
| Wt | : Words of hash calculations |
| <i>temp1</i> | : Temporary register for hash calculations |
| <i>temp2</i> | : Temporary register for hash calculations |

1. INTRODUCTION

1.1 GENERAL INTRODUCTION

Cryptography refers to the data encryption and decryption. Encryption is the process of translating plain text data into something that appears to be random and meaningless (ciphertext), while decryption is the process of converting ciphertext back to plaintext.

The aim of every encryption algorithm is to make a difficult ciphertext to be decrypted without using the key. If a truly good encryption algorithm was used, then there is no technique significantly better than methodically trying every possible key. For algorithm like that, the longer the key, the more difficult to be decrypted a part of the ciphertext without having the key. The problem is if the message contains confidential information, then the message could be intercepted and read by an eavesdropper. That's meaning, the scamper, as usual named as Eve, might be able to modify the message during transmission in such a way that the legal recipient does not detect the manipulation. One objective of cryptography is to provide methods for preventing such attacks [1].

There are two types of keys that are used in cryptography, public and private. The public key is as the name suggests public, where the it is made available to everyone via a publicly accessible repository or directory. On the other hand, the private key must remain confidential to its respective owner. The keys pair is mathematically related, so whatever data encrypting with a public key may only be decrypted by its corresponding private key and vice versa.

There are two types of encryption, symmetric and asymmetric encryption. The only difference between them is the symmetrical encryption uses a single key for both encryption and decryption, while asymmetrical encryption uses a pair of public key and private key to encrypt and decrypt messages.

The process of encryption is used by militaries and governments to simplify secret communication for a long time. The encryption is now generally used in protecting information within many kinds of normal people systems; therefore, cryptography plays an important role in the security of data. It enables to store critical information or sending it across insecure networks so that unauthorized persons cannot read it [2].

The algorithms that using in this thesis are: advanced encryption standard and secure hash algorithms. Secure hash algorithm was used as a key that entered to the AES algorithm. The advantage of combing the AES and SHA algorithms is to make the SHA two way cryptography instead of one way cryptography. The software implementation of the proposed algorithm was implemented via labview 2013 64bits and Matlab 2012a. While the hardware implementation was implemented using a FPGA card called Spartan 3AN 700E.

The cryptographic algorithms which implemented in hardware were more physically secure as they cannot easily be modified or read by an outside attacker. Furthermore, the hardware implementations have a chance of running basically faster than software implementations.

1.2 LITERATURE SURVEY

This section reviews some of the major works available in the literature on AES and hash functions (SHA-1, SHA-2) algorithms. The reviews include their scopes and the ways to improve these algorithms to be more secure and unbreakable. The configuration of the hardware implementations on FPGAs that was introduced:

A.M. Deshpande, M.S. Deshpande and ET. Al in 2009 [3] proposed FPGA implementation of AES based on the Very High Speed Integrated Circuit Hardware Description language (VHDL). ModelSim SE PLUS 5.7g software is used for the simulation and optimization of the synthesizable VHDL code. Synthesizing and implementation of the code is carried out on Xilinx - Project Navigator, ISE 8.2i suite. This paper proposes a method to integrate the AES encrypter and the AES decrypter. The proposed method made a very low-complexity architecture, especially in saving the hardware resource in implementing the AES (Inv) Sub Bytes module and (Inv) Mix columns module. The architecture can still deliver a high data rate in both encryption/decryption operations.

P.B. Ghewari, J. K. Patil and ET. Al in 2010 [4] proposed the implementation of encryption and decryption of AES algorithm with respect to FPGA and Very High Speed Integrated Circuit Hardware Description language (VHDL). Most effective and a better VHDL code developed for the implementation of both, 128 bits data encryption and decryption process. Xilinx ISE 8.1 software is utilized for computer simulation. Every program is examined with some of vectors provided by NIST and the output results were typical with lower delay.

W.Stallings in 2011 [5] given a functional overview of both the basics and workout with regards to cryptography and network security at the beginning of the book, the essential problems related to system security capacity are investigated through giving an instructional exercise and study of cryptography and the network security innovation. The end part of the book includes the act of the network security and the practical applications that have been actualized and used to give the network security.

R.Patel and N.Chaudhary in 2012 [6] was related to hash function, which is a function that takes a random block of data and returns a fixed size bits string called "message digest" this mean any change to the information will change the achieved value. One of the application of the message digest is the digital signature for online transactions. The proposed work tested the robustness of different message digest with regard to digital signature. The repeat sequences in the digest made the robustness of the message digest weak, as well as it may lead to same message digest for different message blocks.

M.Pitchaiah, P.Daniel et al. in 2012 [7] proposed an implementation of AES (128 bits) encryption and decryption by utilizing the Rijndael algorithm. The proposed work was made using Verilog code which implemented based on FPGA card. The AES algorithm is collected from three main parts: cipher, inverse cipher and key expansion. That made cipher converts data to an ambiguous form which called plaintext, furthermore, key expansion produce a Key schedule that is used in cipher and inverse cipher execution.

P.Gehlot, R.Sharma and ET. Al in 2012 [8] presented a comparison between implementing the modules of AES algorithm with different families of FPGA platforms. This methodology uses VHDL to implement the modules in terms of Delay and Frequency. The implementation based on Xilinx – 6.1 xst software and there delay calculations and have been done on FPGA families which are Spartan2, Spartan3 and Virtex2.

R.Ibrahim, R.Kadhim et al. (2015) [9] Presented an implementation of a secure hash algorithm using LabView software. The proposed algorithm of SHA-1 was implemented by LabView software from entering a string, padding, SHA-1 core, and message digest to produce 160 bits hash code for any plaintext. Moreover, an implementation of secure hash algorithm SHA-2 using labview software had been presented. However, the algorithm of SHA-2 proposed in the work is

implemented by LabVIEW software by first entering: string, padding, SHA-2 core, and message digest to produce 256 bits hash code for any plaintext entered.

In **S.Paddhan, S.Shelke et al. (2015) [10]** a modern scheme was made to develop the security of wireless sensor network (WSN) gateway, in WSN the security is a challenging task, due to its limitation towards the low latency in processing, speed and power. Previous results show that the proposed scheme does not have application dependency and have potential in integrated with any application of wireless sensor network. However, the advanced encryption standards (AES) is the technique used in encryption algorithm and sleep scheduler for key management combined and operated with the same key size to provide node authentication and secure key exchange.

In this thesis the AES was combined with the SHA. The proposed system software was built via Labview 2013 (64 bits) and Matlab 2012a. After that the proposed system was implemented on an FPGA card (Spartan 3AN) that was connected via JTAG cable to HP laptop EliteBook corie5 (64 bits) 2.5 GHz CPU..

1.3 AIM OF THESIS

The aim of the proposed system is to design an encryption algorithm that combines the AES and hash algorithms together by using LabView software in order to achieve confidentiality. The SHA was used as a key to the AES and that led to making the SHA two way cryptography like the AES instead of one way (encryption only).

These are steps implemented as the following:

1. Build the proposed system in software and shows the easiness of the implementation using Labview.
2. Build the proposed system in hardware and compare the results using the FPGA card to take the advantage of it in various applications.

1.4 THESIS CONTENT

1. Contains a general introduction, the literature survey and the aim of the thesis.
2. Includes introduction to the Advanced Encryption Standard and hash functions by discussing their requirements and some of their attacks. This chapter also includes the decryption of AES and the relationship between the AES and hash functions.
3. Presents the implementation of the proposed system.
4. This chapter presents the results and the simulation test of all algorithms that were implemented in chapter three.
5. Contains the conclusions and the suggestions for future works.

2. AES AND HASH FUNCTION ALGORITHMS

2.1 INTRODUCTION

This chapter focuses on cryptography techniques. The techniques are: advanced encryption standard algorithm and cryptographic hash functions. Some of these attacks, block cipher modes of operation, and Field Programmable Gate Array were discussed. Finally, the measure of the cipher randomness by applying cryptography strength tests on the proposed algorithm of our system.

2.2 CRYPTOGRAPHY

Cryptography is the exploration of transmitting messages vigorously and safely from a transmitter to a beneficiary. The point is to encode the message in a way that a meddler couldn't have the capacity to comprehend its substance. By reproducing the essentials of cryptographic situation, and assuming that there are two clients Alice and Bob who need to exchange messages between each other solidly, so that the spy, Eve, couldn't get the information that transmitted. They will utilize the cryptography exploration of guarding messages secure and safe. By observing Figure 2.1, if Alice needs to send the plaintext (message) to Bob, she will utilize to some encryption strategies to change this message to ciphertext. This ciphertext ought to be incoherent to any outsider, additionally ready to be decrypted once it has been gotten by Bob [11].

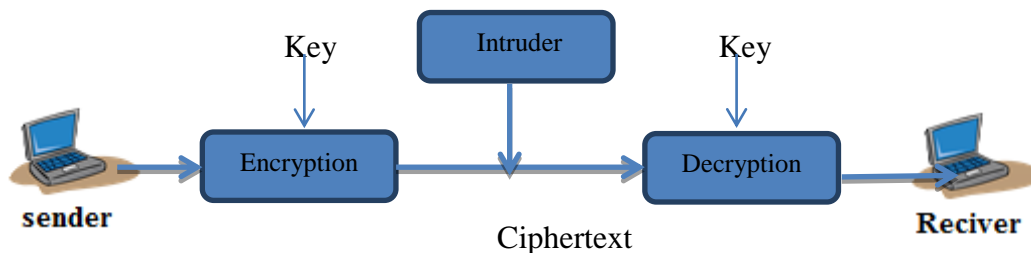


Figure 2.1: Encryption and decryption.

In this area, the motivation behind cryptography and how to utilize it in the cryptosystems is depicted and security necessities and ideas, including authentication, integrity, digital signature

and confidentiality or privacy. Cryptography is not just worried with encryption and unscrambling messages, likewise it is utilized to take care of issues in genuine which require data security.

Two types of cryptography are used:

1. One way cryptography:

There are two parts of cryptography functions, one way function and two way function.

One way function, means a function that it easy to compute in every input, but hard to invert giving a random image of the input. In easy and hard it means to be understood in the computational complexity theory. Computational complexity theory is a subfield of theoretical computer science one of primary goals is to classify and compare the practical difficulty of solving problems about finite computational objects [12].

One way encryption or hashing function $H(M)$ can be used to map data of arbitrary size to data of fixed size. In a simple way, hash function $H(M)$ can calculate any length of data (M) to a fixed length result h . Given M , it is easy to compute h , given h , it is hard to compute M such that $H(M) = h$, Given M , it is hard to find another message, M' , such that $H(M) = H(M')$ [13].

Furthermore, the results of one way encryption or hashing after encrypted cannot be decrypted back to original data. The useful use of hashing generally is used to store password or to detect duplicated records in a large file [14].

2. Two way cryptography:

Two way cryptography refers that the input encrypted can be decrypted and returns to its' original value. DES and AES are a two way cryptography. Data encryption standard (DES) encrypts data in 64-bits blocks by using a 56-bits key. The DES algorithm transforms 64-bits input in a series of steps into a 64-bits output. Same steps with the same key are used to reverse the encryption [15]. Advanced encryption standard is described in details in the next sections

2.2.1 Authentication

Communication requires security that any individual communicating can identify the identity of which and whom they communicate and transmit the message. The basic definition of authentication is the procedure of checking the identity of a person or entity. By returning to the basic scenario of cryptography, which is about Alice and Bob to know the fundamental of authentication, which is important to consider that Bob wants to be sure that the message he received is sent by Alice only [15].

There are two forms of authentication in cryptography: entity authentication and data-origin authentication. The first type is which expressed by the term identification that is interested in the identity of the parties included within the network [16]. Data-origin authentication concentrates on attaching the information about the origin of data with the data transferred, such as the creator and the time of creation [11] [17]. Identification is very important when logging in to any account that belongs to the user to identify him/herself, typing a username only is not sufficient to prove that account belong to the right user because there are many users who are interested in the same name, so there is a need to password to prove the account verification [18], as shown in Figure 2.2.

The diagram illustrates a two-step authentication process. **Step 1** involves entering credentials: 'Email/Username' (your-username) and 'Password' (*****), followed by a 'Login' button. **Step 2** involves entering a verification code (123456) and a 'Verify' button. An SMS icon indicates a code received of 123456.

Figure 2.2: Example on an Authentication

2.2.2 Integrity

Protecting data from being altered by not common intruders called Integrity. Information has a value only if it is correct and could be invaluable if it has been altered. For example, if a person sends an online money transfer of \$500 but the data was altered in such a way that the person actually sent \$50,000, it could be very costly for the sender.

In a similar format, for an information recipient, with the essential situation considers the collector is Bob, must make certain that the information sent by the maker (Alice) is the very same information got with no change [19].

Some methods are usually used to protect data integrity like "hashing", which mean received data and make a comparison between the received hash and the hash of the original message. According to this the hash of the original data must be given to the receiver in a secure way, as shown in Figure 2.3.

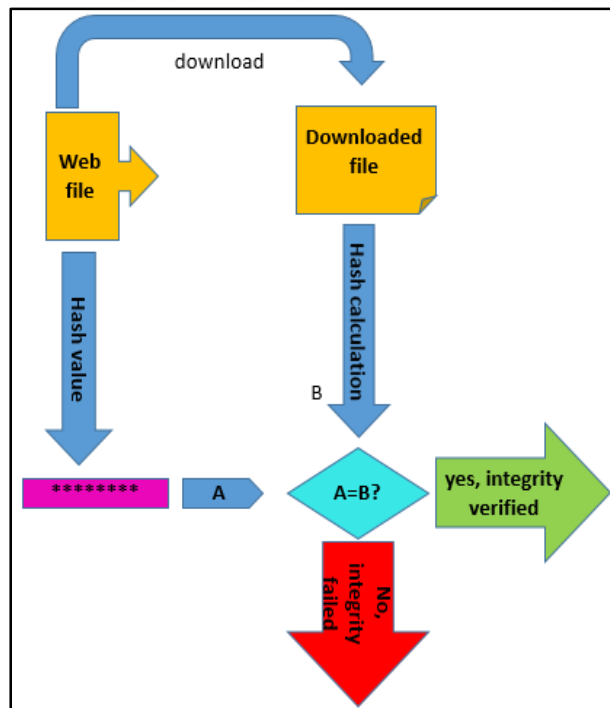


Figure 2.3: Message integrity

2.2.3 Digital Signature

The idea of the signature is to present the reality and the virtual advanced world is the same in the genuine correspondence. For example, any paper in the organization conveys a choice from the director goes to the customers ought to convey a signature of the supervisor to demonstrate that choice is solid and not produced [1]. Digital signature used to spread a message in plaintext frame when the collectors must check and distinguish the message transmitter. The Signing message doesn't change the message, it essentially produces a computerized signature code string that can either reach out to the message or transmitted independently [20]. A digital signature is a short section of information that is encoded with the sender's private key, as appeared in Figure 2.4. Unscrambling the signature information utilizing the sender's public key demonstrates that the information was scrambled by the transmitter or by somebody who had admittance to the transmitter's private key [11] [15].

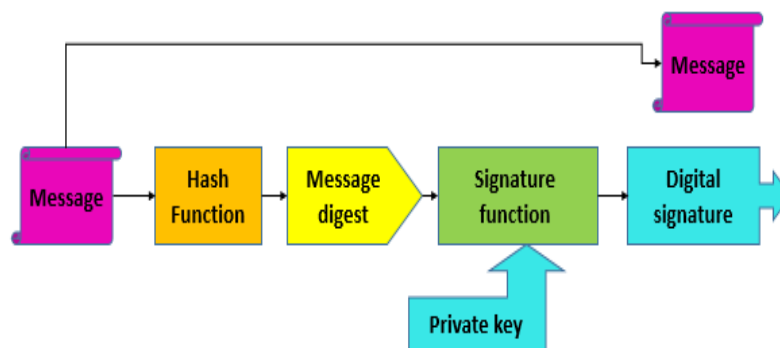


Figure 2.4: Digital Signature Creation

2.2.4 Confidentiality

Confidentiality is described as an office which is used to protect the content of data from all messages but persons approved to get it. Protection is a period indistinguishable with Confidentiality and mystery. There are fluctuated strategies to getting Confidentiality, reaching out from framework security to registering calculations which reduce data unfathomable [21], as shown in Figure 2.5.

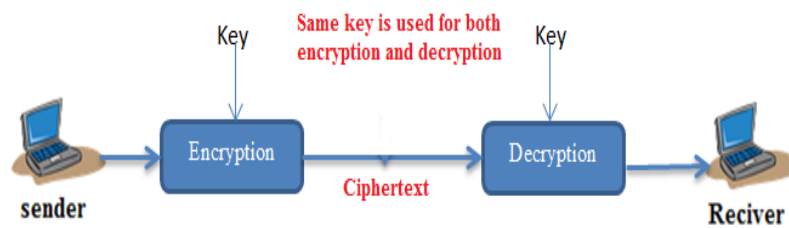


Figure 2.5: Message Confidentiality achieved by encrypt/decrypt

Confidentiality is generally proportionate to privacy. Measures embraced to guarantee confidentiality is intended to keep touchy data from contacting the wrong users, while ensuring that the correct users can in actuality get it: Access must be confined to those approved to see the information being referred to. In addition, it is normal for information to be sorted by the sum and kind of harm that should be to fall into unintended hands. Pretty much stringent measures can then be actualized by those classifications [22].

Confidentiality and integrity are achieved by making the hash functions as an input password to the AES case structure and this can be seen in details in the software implementation of the AES in chapter three.

2.3 ADVANCED ENCRYPTION STANDARD (AES)

Advanced Encryption Standard (AES) is the current standard for secret key encryption, it was created by two Belgian cryptographers, Vincent Rijmen and Joan Daemen. A standardized version of the algorithm used by the Federal Information Processing Standard 197 called Rijndael for the Advanced Encryption Standard. A combination of exclusive-OR operations (XOR), octet substitution with an S-box, row and column rotations and a MixColumn used by the algorithm. It was easy to implement and could run in a reasonable amount of time on a regular computer [23].

AES is a United States encryption standard known in Federal Information Processing Standard "FIPS" 192. AES is a harmonic encryption algorithm handling data in block of 128 bits [24]. AES is harmonic since the same key is used for encryption and the decryption [25]. The key is the only mystery important to keep for security. AES is configured to utilize various key lengths it has that 3 lengths defined by the standard and the resulting algorithms are named AES-128, AES-192 and AES-256 respectively to indicate the length in bits of the key. Figure 2.6 shows a flow chart of the Advanced Encryption Standard.

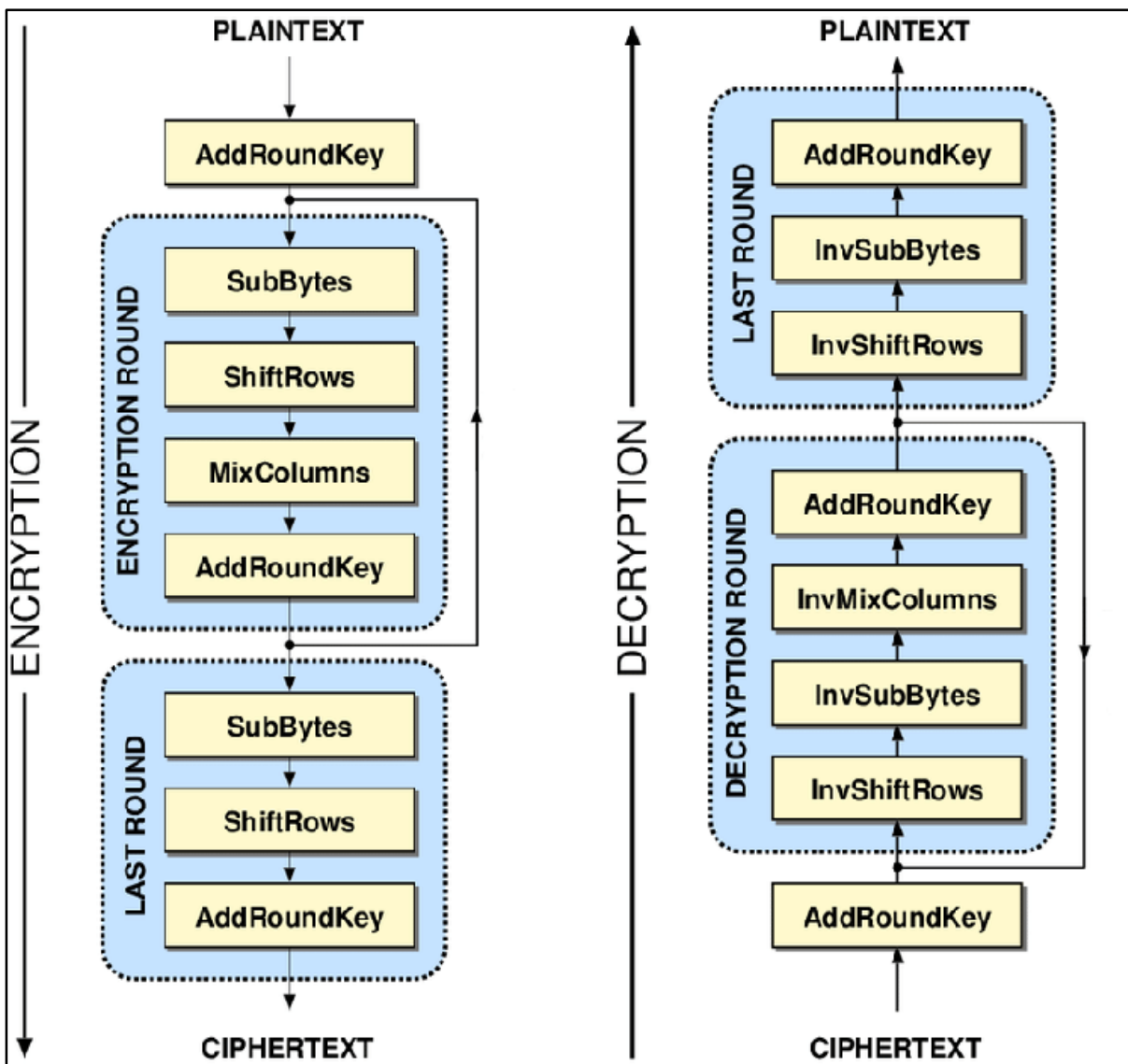


Figure 2.6: General structure of AES algorithm

2.3.1 AES Encryption

Figure 2.6 demonstrates the common structure of the AES encryption process. The cipher picks a plaintext block size of 128 bits (16 bytes) and the key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is alluded to as AES-128, AES-192, or AES-256, contingent onto the key length [15].

Block cipher consists of N rounds, where the number of rounds depends on the key length: 10 rounds for a 16-bytes key, 12 rounds for a 24-bytes key, and 14 rounds for a 32-bytes key. The principal rounds comprise of four distinct transformation functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey, which are described subsequently. The last round contains just three transformations, and there is an initial single transformation (AddRoundKey) before the main round, which can be viewed as Round 0. Each round key fill in as one of the contributions to the AddRoundKey transformation in each round. The algorithm starts with an Add round key stage.

There are four steps in AES encryption, Sub Bytes, Shift Rows, Mixing Columns and Add Round Key which are discussed in this section.

a. Sub Bytes Transformation

The Sub Bytes transformation is a nonlinear bytes substitution that acts on every bytes of the state separately to produce a new bytes value using an S-box substitution table, this is called the substitution layer. As appeared in Figure 2.7 ,the substitution layer is based on the S- box or (the AES substitution table) which is invertible and is achieved by initializing the table in ascending sequence row by row, after that, a structure of the following transformations is taken:

1. First, take the multiplicative inverse in GF (2⁸) the value of {00} is mapped to itself.
2. Second, applying an affine-over GF (2)-transformation defined by:

$$Y_i = x_i \oplus x_{(i+4) \bmod 8} \oplus x_{(i+5) \bmod 8} \oplus x_{(i+6) \bmod 8} \oplus x_{(i+7) \bmod 8} \oplus c_i \quad (2.1)$$

For $0 \leq i < 8$ where:

x_i is the i^{th} bits of the input bytes, y_i is the i^{th} bits of the output bytes and c_i is the i^{th} bits of a bytes c with the value $\{63\}$.

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

(2.2)

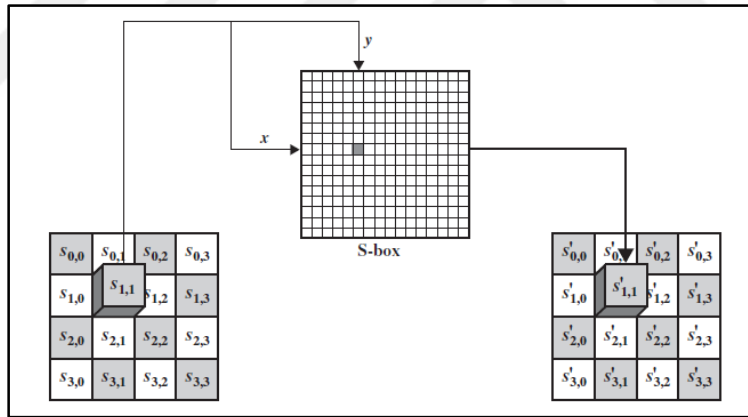


Figure 2.7: Sub-Bytes transformation [15]

S-box and inverted S-box that used in this section are found in Appendix A. Also $GF(2^8)$ is shown with more details in Appendix D.

b. Shift Rows Transformation

In the step of ShiftRows transformation, the bytes in last three rows of the state are cyclically shifted over different numbers of bytes (offsets), as shown in Figure 2.8. Row 0 is not shifted;

row 1 is shifted over C1 bytes, row 2 over C2 bytes and row 3 over C3 bytes. The shift offsets C1, C2 and C3 depend on the block length Nb.

Table 2.1: Shift offsets for different block lengths

| Nb | C1 | C2 | C3 |
|----|----|----|----|
| 4 | 1 | 2 | 3 |
| 6 | 1 | 2 | 3 |
| 8 | 1 | 3 | 4 |

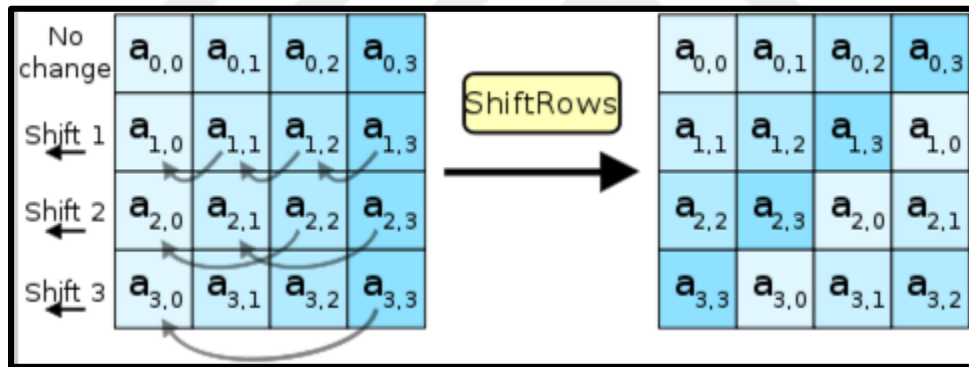


Figure 2.8: Shift Rows cyclically shift the last three rows in the State

c. Mix Columns Transformation

The mix Column transformation along the shift rows step works on the state column by column by handling each column as a four term polynomial. Every bytes of a column is charted into a new value that is a function of all four bytes in that column. The transformation can be introduced by the following matrix:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0}s_{0,1}s_{0,2}s_{0,3} \\ s_{1,0}s_{1,1}s_{1,2}s_{1,3} \\ s_{2,0}s_{2,1}s_{2,2}s_{2,3} \\ s_{3,0}s_{3,1}s_{3,2}s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0}s'_{0,1}s'_{0,2}s'_{0,3} \\ s'_{1,0}s'_{1,1}s'_{1,2}s'_{1,3} \\ s'_{2,0}s'_{2,1}s'_{2,2}s'_{2,3} \\ s'_{3,0}s'_{3,1}s'_{3,2}s'_{3,3} \end{bmatrix} \quad (2.3)$$

Each or every element in the product matrix is the summation of products of elements of one row and one column. In this situation, the single additions and multiplications are performed in $GF(2^8)$. The MixColumns transformation on an individual column of State can be implemented as:

$$\left. \begin{aligned} S'_{0,j} &= (2 \cdot s_{0,j}) + (3 \cdot s_{1,j}) + s_{2,j} + s_{3,j} \\ S'_{1,j} &= s_{0,j} + (2 \cdot s_{1,j}) + (3 \cdot s_{2,j}) + s_{3,j} \\ S'_{2,j} &= s_{0,j} + s_{1,j} + (2 \cdot s_{2,j}) + (3 \cdot s_{3,j}) \\ S'_{3,j} &= (3 \cdot s_{0,j}) + s_{1,j} + s_{2,j} + (2 \cdot s_{3,j}) \end{aligned} \right\} \quad (2.4)$$

d. Add Round Key

In the forward add round key transformation, a round key is added to the state by a simple bitwise XOR operation. The operation is seen as a column wise operation between the bytes of a state segment and single word of the round key; it can likewise be seen as a bytes-level operation.

e. Key Expansion

The AES uses a cipher key whose length is 128, 192 or 256 bits. This cipher key is expanded into 10, 12, or 14 round keys respectively, using the Key Expansion algorithm, where the length of each round key is 128 bits. This Key Expansion algorithm depends only on the cipher key. Since it independent of the processed data, it can be (and typically is) executed prior to the encryption/decryption phase.

The AES calculation takes the Cipher Key, K (key length) and plays out a Key Expansion routine to produce a key timetable. The Key Expansion creates an aggregate of N_b ($N_r + 1$) words: the calculation requires an underlying arrangement of N_b (number of words in an AES block) words and each of the N_r (Number of Rounds) rounds require N_b expressions of key information. The subsequent key calendar comprises of a straight exhibits of 4-bytes words,

signified $[WI]$, with I in the range $0 < I < Nb (Nr + 1)$. The extension of the information entered into the key timetable continues as indicated by the pseudo code.

Sub word is a function that picks 4 bytes input word and applies the S-box to each of the 4 bytes in order to produce an output word. The function Rot Word picks a word $[a_0, a_1, a_2, a_3]$ as information, plays out a periodic change and returns the word $[a_1, a_2, a_3, a_0]$. Rcon $[I]$ (Rcon is the exponentiation of 2 to a user given value), contains the qualities given by $[x^{i-1}, \{00\}, \{00\}, \{00\}]$, with x^{i-1} being forces of (x is signified as $\{02\}$) in the field $GF(28)$. The main N_k terms of the extended key are expanded with the cipher key. After each word, $w[i]$ is equal to the XOR of the previous word ($w[i-1]$). For the words in positions that are a plenty of N_k , a change is connected to $w[i-1]$ before the XOR, withdrawn by a XOR with a round consistent, Rcon $[i]$ as appeared in Figure 2.9. This change comprises of a cyclic move of the bytes in a word (RotWord), trailed by the utilization of a table query to all Four bytes of the word (SubWord). Note that the Key Expansion routine for 256-piece Cipher Keys ($N_k = 8$) is marginally not quite the same as for 128-and 192-piece Cipher Keys. In the event that $N_k = 8$ and $i-4$ is a different of N_k , then SubWord () applies to $w[i-1]$ before the XOR.

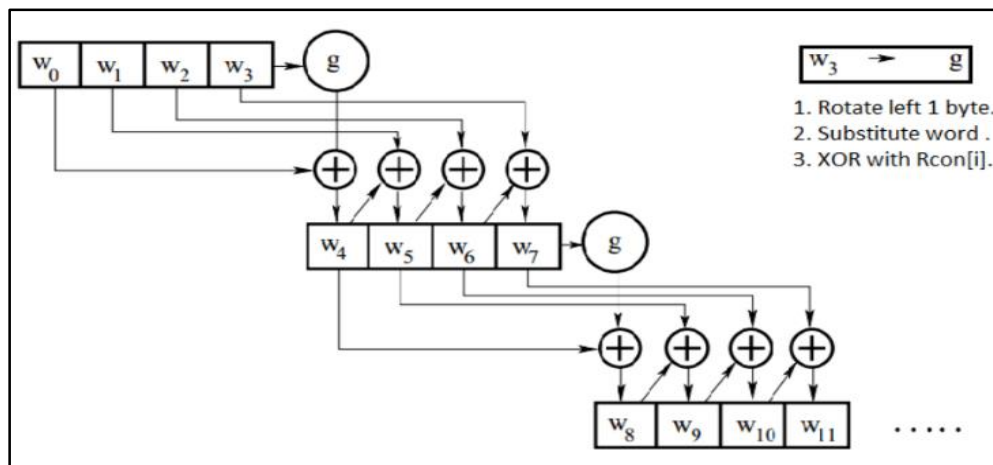


Figure 2.9: General Block Diagram of the Key Expansion [26]

The process involved is to rotate cyclic left by 1 bytes and then substitute rotated word with the S-Box as shown in Table (A.1) in Appendix A. The substituted word must be XORed with, round constant Rcon $[i]$. The word of the round constant Rcon $[i]$ include the values given by $[x^{i-1}, \{00\}, \{00\}, \{00\}]$, with x^{i-1} being powers of x (x is denoted as $\{02\}$) in the field GF

(2^8) , where (i) here starts at 1 not 0. For w4 value of i for $Rcon[i]$ will be 1 and is increased in each round [26].

2.3.2 AES Decryption

In the decryption mode, the operations are backward request contrasted with their request in encryption mode. It begins with an underlying round, trailed by 9 emphasis of a backwards typical round and closes with an Add Round Key. A backwards ordinary round comprises of the accompanying operations in a specific order: Add Round Key, Inverse Mix Columns, Inverse Shift Rows, and Inverse Sub Bytes. An underlying round is a reverse typical round without the Inverse Mix Columns. [27].

a. Inverse Shift Rows Transformation

Opposite Shift rows is the reverse of the Shift rows transformation. The bytes which are in the last three rows of the state are systematically moved over many quantities of bytes. The main row, $R = 0$, is not moved. The last three rows are systematically moved by $[Nb - \text{shift}(r, Nb)]$ bytes, where the move bits shift (r, Nb) depends on the line number as shown in Figure 2.10.

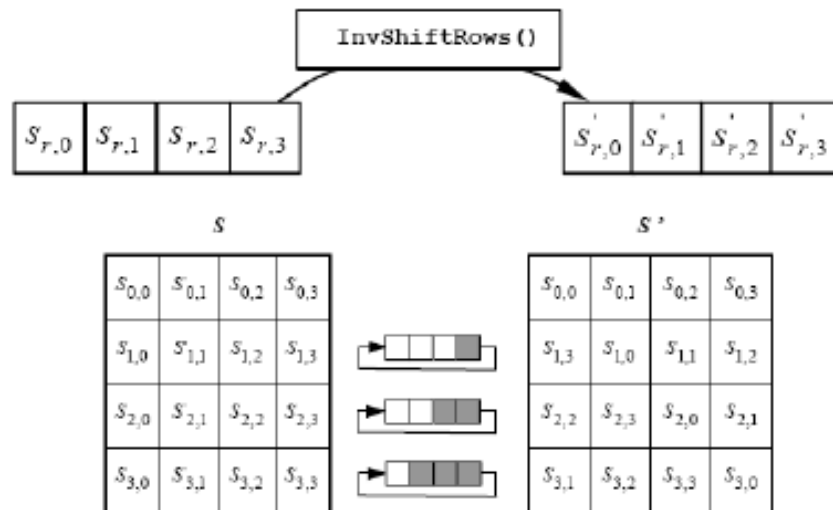


Figure 2.10: Inverse Shift Rows Transformation

b. Inverse Sub Bytes Transformation

Inverse Sub Bytes is the reverse of the bytes substitution transformation, in which the inverse Sbox is applied to each bytes of the State. This is obtained by applying the inverse of the affine transformation followed by taking the multiplicative inverse in $GF(2^8)$.

c. Inverse Mix Columns Transformation

Inverse Mix Columns is the reverse of the Mix Columns transformation. Inverse Mix Columns works on the state column by column, handling every column as a four term polynomial. The columns are assumed as polynomials over $GF(2^8)$ and multiplied modulo $x^4 + 1$ with a constant polynomial $a^{-1}(x)$, given by $a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}$, this can be written as a matrix multiplication. Let As a result of this multiplication, the four bytes in a column are exchange by the matrix bellow:

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} \dots \quad (2.5)$$

$$\left. \begin{aligned} S'_{0,c} &= (\{0e\} \cdot s_{0,c}) + (\{0b\} \cdot s_{1,c}) + (\{0d\} \cdot s_{2,c}) + (\{09\} \cdot s_{3,c}) \\ S'_{1,c} &= (\{09\} \cdot s_{0,c}) + (\{0e\} \cdot s_{1,c}) + (\{0b\} \cdot s_{2,c}) + (\{0d\} \cdot s_{3,c}) \\ S'_{2,c} &= (\{0d\} \cdot s_{0,c}) + (\{09\} \cdot s_{1,c}) + (\{0e\} \cdot s_{2,c}) + (\{0b\} \cdot s_{3,c}) \\ S'_{3,c} &= (\{0b\} \cdot s_{0,c}) + (\{0d\} \cdot s_{1,c}) + (\{09\} \cdot s_{2,c}) + (\{0e\} \cdot s_{3,c}) \end{aligned} \right\} \dots \quad (2.6)$$

d. Inverse of Add Round Key Transformation

Add Round Key own inverse since it only involves an application of the XOR operation. The inverse cipher transformations differ from Cipher transformation, while the form of the key schedules for encryption and decryption stills the same. However, many properties of the AES algorithm allows for the Inverse Cipher that has exactly same sequence of transformations as the Cipher (with the transformations exchanged by their inverses). This is completed with an alteration in the key schedule.

2.4 BLOCK CIPHER MODES OF OPERATION

Five modes of operation are used when applying block ciphers in a variety of applications. These modes are: electronic codebook mode, cipher block chaining mode, cipher feedback mode, output feedback mode and counter mode. These modes are explained in this section [15].

2.4.1 Electronic Codebook Mode (ECB)

Electronic codebook is the simplest mode between the five modes. Figure 2.11 shows the plan where a block of plaintext (which is a similar size for every situation) is encrypted with a comparable key K . The expression codebook is utilized in light of the fact that, for a specific key, there is a new ciphertext for each block of a plaintext. In this manner, a huge codebook could be imagined, where there is a passage for each possible plaintext design by demonstrating the related ciphertext of it. If the message is more in length in comparison with the block length, the strategy would be, breaking the message into squares of the required length and padding the last block if needed. Likewise, encryption and decryption could be performed one block at any given moment, dependably utilizing a similar key .

The ECB technique is perfect for little measures of information. For example, an encryption key in any case, for longer messages if the same plaintext block was seen more than once, then the same ciphertext is delivered. This may help the attackers.

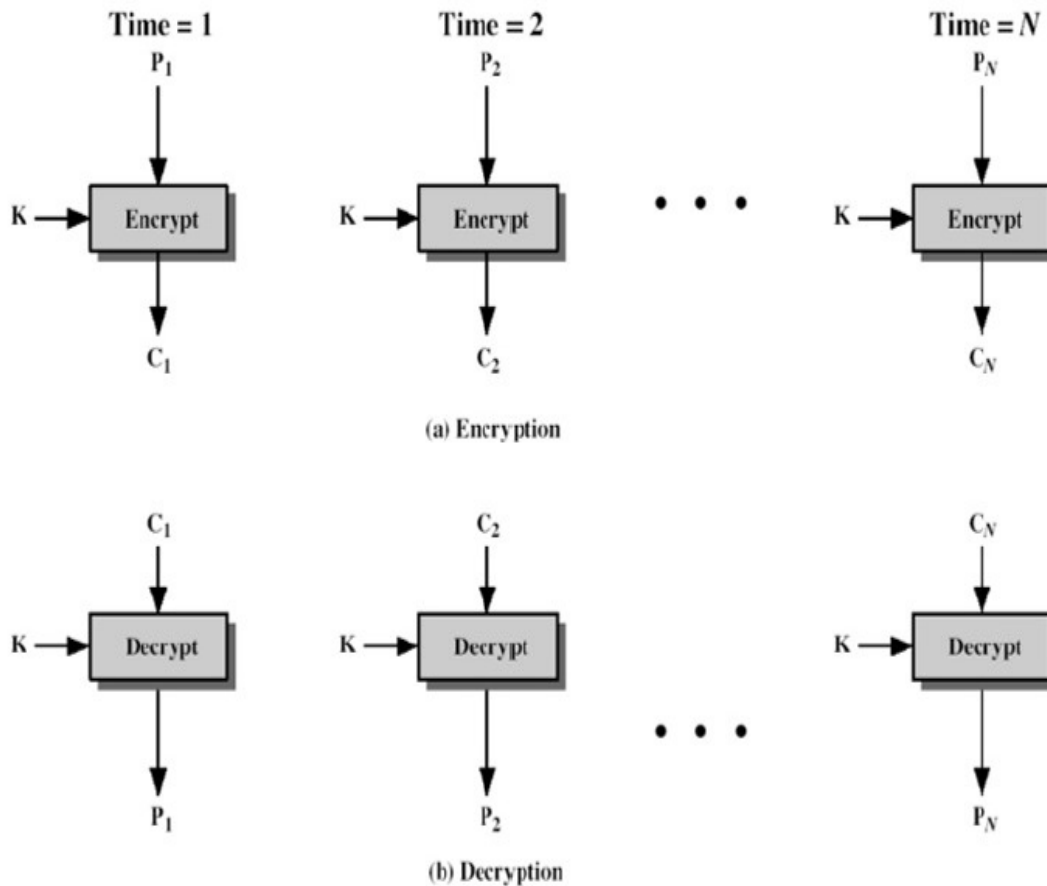


Figure 2.11: ECB mode

2.4.2 Cipher Block Chaining (CBC) Mode

The same plaintext blocks produce distinctive ciphertext blocks. Cipher Block Chaining permits this by XORing each plaintext with the ciphertext from the past round (the first round utilizing an Initialization Vector (IV) (As sometime recently, a similar key is utilized for each block), Encrypting functions shown in the Figure 2.12 (a). Decrypting functions as appeared in the Figure 2.12 (b) in light of the properties of the XOR operation, i.e. $IV \text{ xor } IV \text{ xor } P = P$ where IV is the Initialization Vector and P is the plaintext. Clearly, the IV should be known by both sender and got and it ought to be kept a mystery alongside the key for most extreme security.

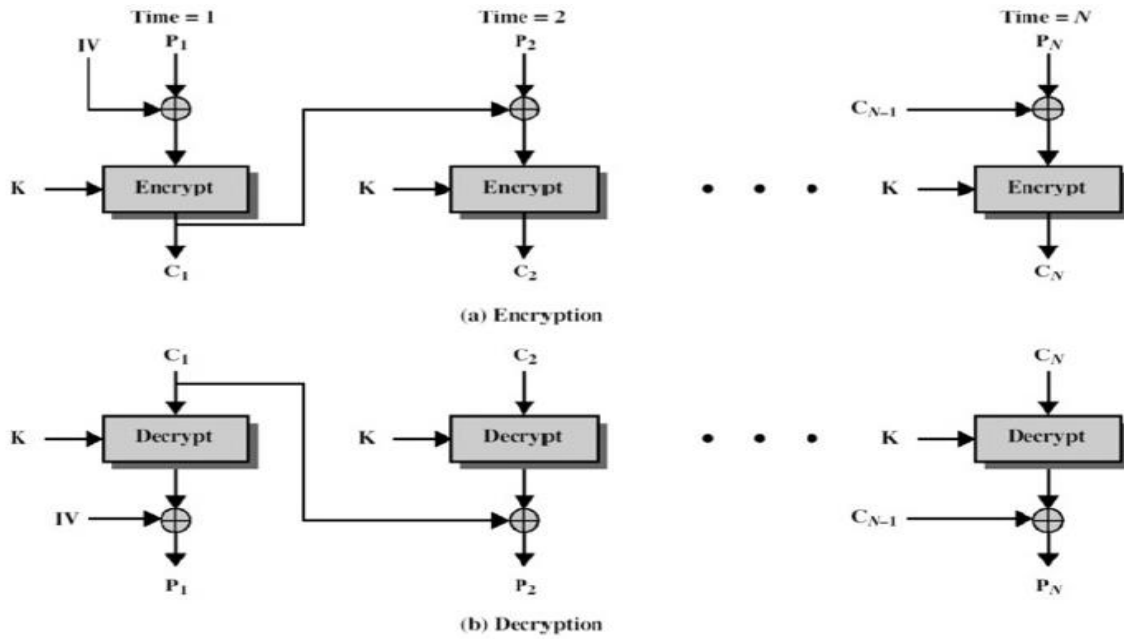


Figure 2.12: CBC Mode.

2.4.3 Cipher Feedback Mode (CFB)

The Cipher Feedback allows a block cipher to be altered into a stream cipher. There is no need to pad a message to be an indispensable number of blocks. Likewise, it could work in real time. Figure (2.13) demonstrates the CFB schema. In this Figure, the unit of conveyance is s bits; a typical esteem is $s = 8$. Like the CBC mode, the units of plaintext are tied together, so that the ciphertext of each plaintext unit is a component of all the previous plaintext (which is part into s bits fragments). The contribution to the encryption task is move enroll break even with long to the square Figure of the algorithm (despite the fact that the graph refers to 64 bits, which is block size utilized by DES, and that could be stretched out to other block sizes, for example, the 128 bits of the AES). Some Initialization Vector (IV) is set at the beginning. The leftmost s bits of the encryption function are XORed with, to begin with a piece of plaintext P_1 to deliver the main unit of ciphertext C_1 which is then transmitted. What's more, the contents of the shift register are shifted left by s bits and C_1 is set in the least significant s bits of the shift register. This process proceeds until all plaintext units are encrypted. Decrypting is comparable

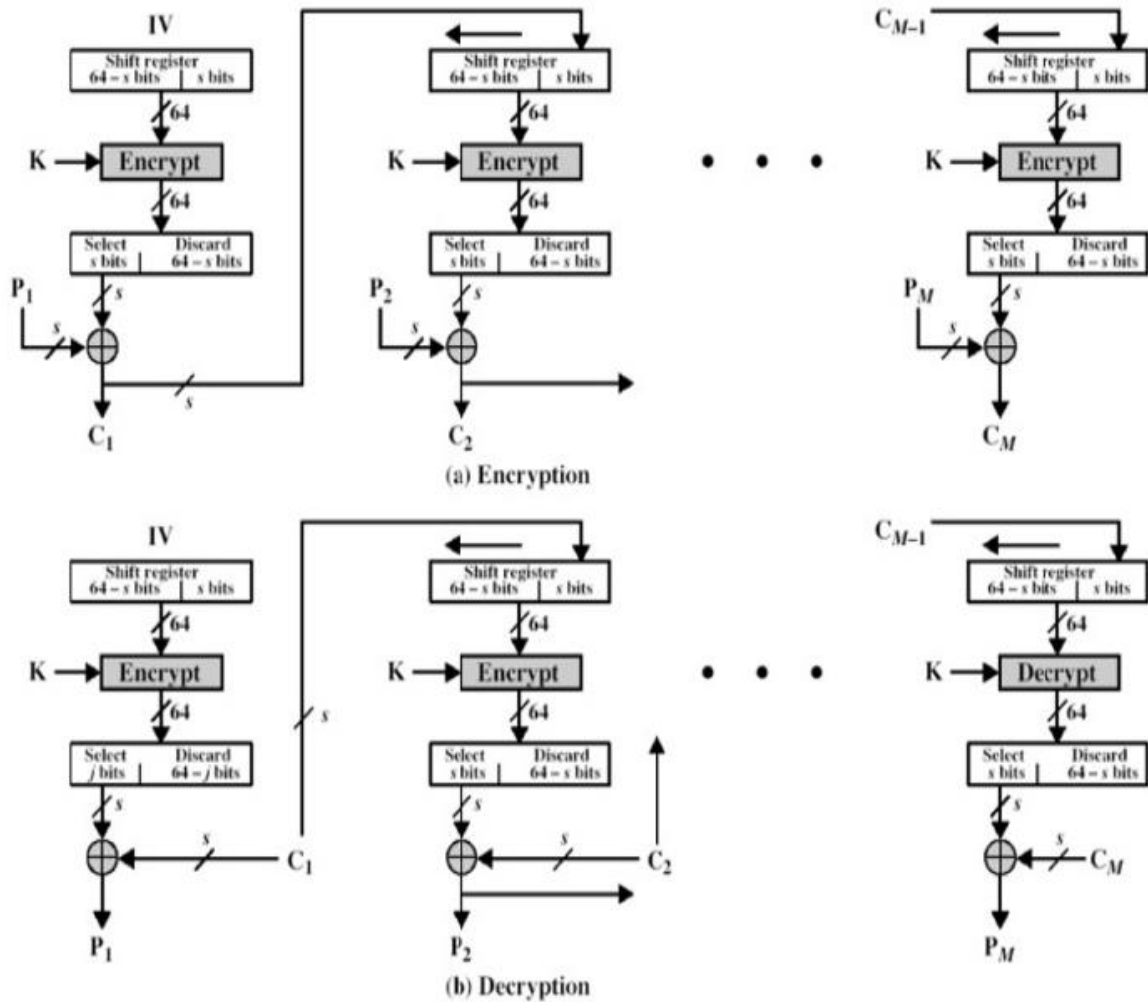


Figure 2.13: CFB Mode.

2.4.4 Output Feedback (OFB) Mode

The Output Feedback Mode is comparative in structure to the CFB mode, as found in Figure (2.14). The yield of the encryption function is fed back to the shift register in OFB, while in CFB the ciphertext unit is fed back to the shift register. One wanted standpoint of the OFB strategy, the bits errors in transmission do not increased. For instance, if the bits error happens in C_1 just the recouped estimation of P_1 is influenced; the achieved plaintext units are not corrupted. With CFB, C_1 likewise fills in as input to the shift register and in this way causes extra defilement downstream.

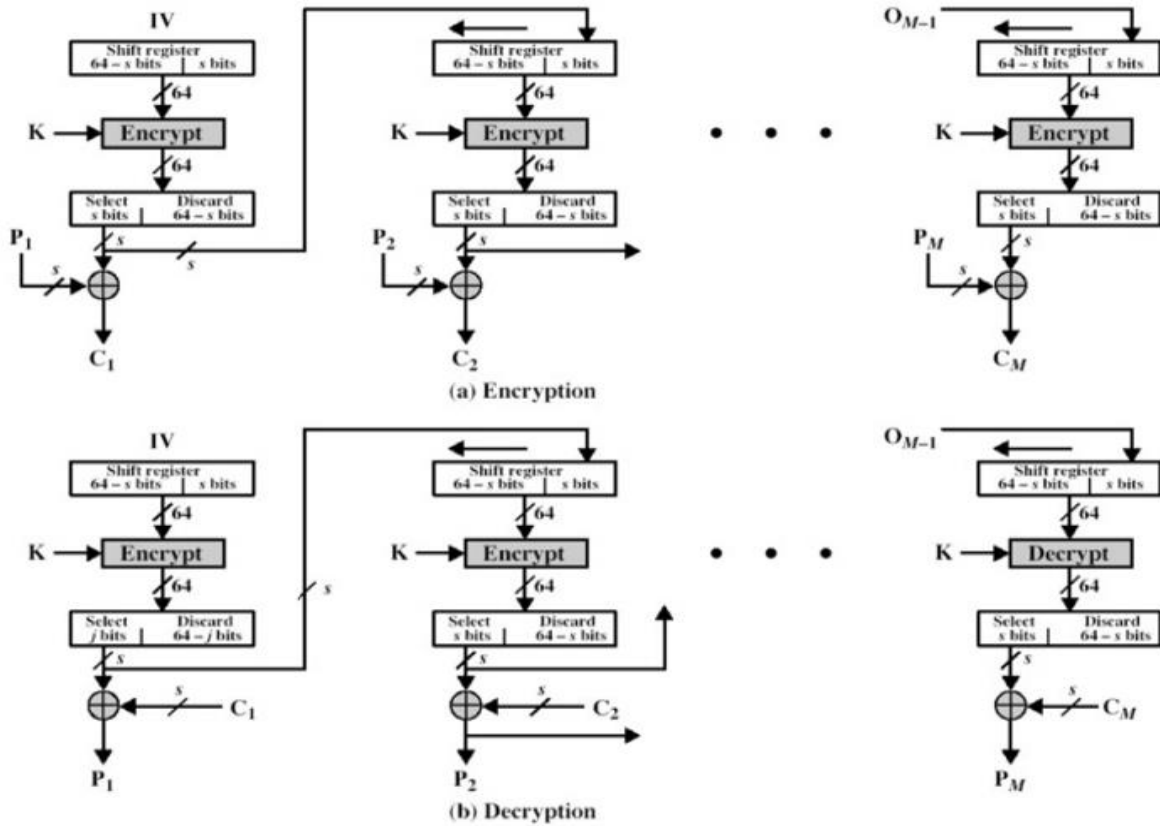


Figure 2.14: Output Feedback (OFB)

2.4.5 Counter (CTR)

Interest in this mode has developed in a great way recently. A counter equivalent to the plaintext block size is utilized. The main necessity expressed in the standard is that the counter value should be distinctive for every plaintext obstruct that is encrypted. Commonly, the CTR is initialized to some value and after that it will be increased by 1 for every resulting square (modulo 2^b where b is the block size), as shown in Figure 2.15. For encryption, the counter is encrypted and then XORed with the plaintext to deliver the ciphertext block; no chaining is found. For decrypting, a similar arrangement of counter values is utilized, with every encrypted counter XORed with a ciphertext block to get the comparing plaintext block. This mode contains various favorable advantages including hardware efficiency, software efficiency, provable security (as in it is in any event as secure as alternate modes previously) and simplicity.

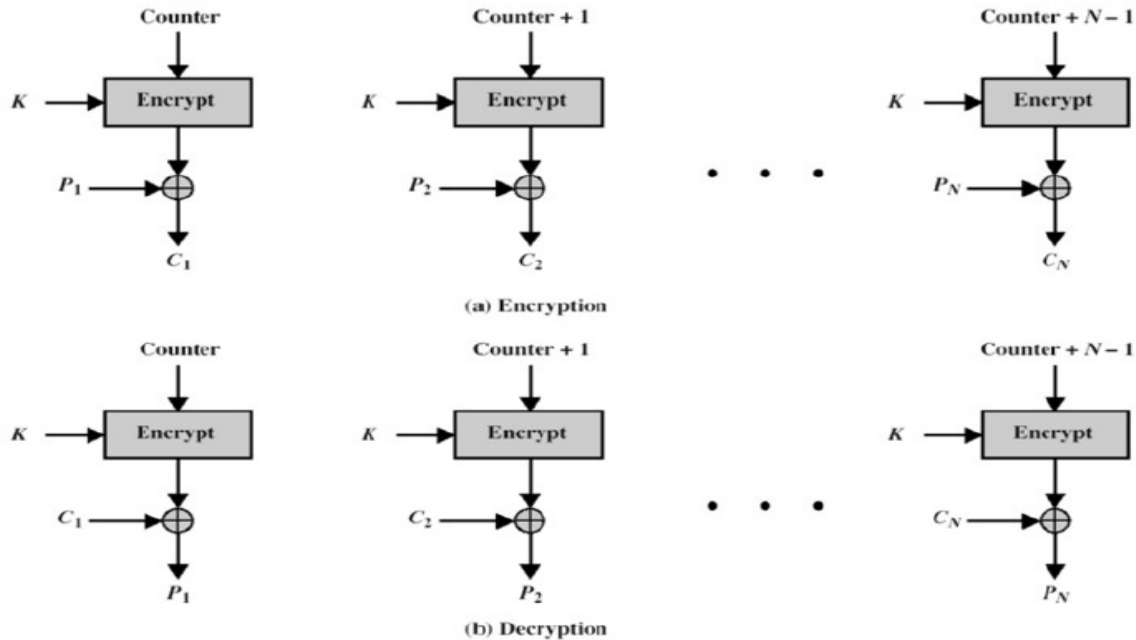


Figure 2.15: Counter (CTR) Mode

2.5 SECURE HASH ALGORITHM (SHA)

Hash functions are functions that comprises an input of arbitrary length to a result with a fixed length [28]. Hashes are used to confirm information and message integrity, password authenticity the fundamental of many other cryptographic systems. Every hash is individual, but always repeatable. The word 'dog' will hash to something that no other word hashes too, but it would always hash to the same thing [29].

2.5.1 Secure Hash Algorithm (SHA-1)

SHA-1 makes a 160-bits (20 bytes) message digest. The main specification of the SHA-1 algorithm was released in 1993 based on the Secure Hash Standard (FIPS PUB 180) by US government standards agency NIST. This version is now often referred to as SHA-0. It was replaced by the revised version, published in 1995 in FIPS PUB 180-1 and commonly pointed to as SHA-1. SHA-1 differs from SHA-0 only by a single bitwise shifting in the message schedule

of its compression function; this was done, according to the national security agency NSA, to correct a flaw in the original algorithm which decreased its cryptographic security [30].

The Secure Hash Algorithm (SHA-1), Merkle-Damgård construction was designed by R. Merkle [31] and I. Damgård[32] .

The vast majority of the hash functions and all the steady hash functions are based on message digest construction as appeared in Figure 2.16.

Message digest construction is basically taken care of in three steps. Initial step is the padding. The motivation behind the padding is to sort the length of message a different of message block length, m . The most normally utilized padding procedure is: a "1" bits taken after series of "0" bits and the bitwise documentation of the message length are added to the message. The quantity of "0" bits added to the message are chosen so that the message length transforms into a various of block length m . Comprehensively the greatest length of the message that can be dealt with by the hash function is $264 - 1$ [33].

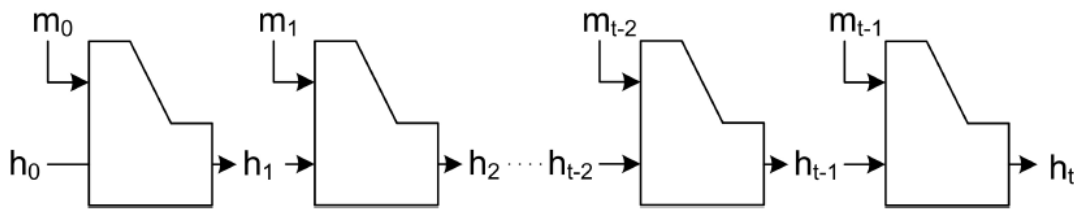


Figure 2.16: Message digests construction [33]

The Second step is isolating the padded message into m bits blocks $m_0, m_1, m_2 \dots m_{t-1}$. Next of this progression, the binding values are recursively settled by utilizing a fixed publicly recognized as an initialization vector (IV), and the message pieces.

$$h_0 = IV \tag{2.7}$$

where $i = 1, 2 \dots, t$
$$h_i = f(h_{i-1}, m_{i-1}) \tag{2.8}$$

Where f is the compression function of the hash. Generally, H is thought to be the message hash value. The vital property of Merkle-Damgård structure is the collision resistance protection of

the compression function [31] [32]. The message digest development expends a wellbeing affirmation that if (f) utilized as a part of hash algorithm collision resistant, then the hash function is collision resistant. Additionally, the impact resistance that assumed the message digest structure likewise saves the preimage resistance and second preimage resistance of (f).

The input is passed to the system must be managed in 512 bits message blocks. The procedure of the SHA-1 algorithm is described as followed:

Step 1: Padding bits:

The message is padded until the length in bits is comparing to 448 modulo 512. Subsequently, the length of the padded message is 64-bitss less a multiple of 512-bitss. Padding is persistently supplementary, regardless of the possibility that the message is now of the desired length, in light of the fact that the message M is isolated into chunks everyone called m [33] [34].

Step 2: Append length:

A message chain of 64-bitss unsigned 64-bits number is added to the bits of the message for the thought processes of security. This chain of message bits covers the length of the first bits of the message before including the padding bits [33] [34].

Step 3: Initialization Vector:

A 160-bits buffer of a message is saved to clamp center and the hash function comes about. This buffer denoted as five 32-bits (A, B, C, D and E) registers. The registers qualities are set up with the accompanying 32-bits constant values in hexadecimal shape: [33] [34]

A: 67452301

B: EFCDAB89

C: 98BADCFE

D: 10325476

E: C3D2E1F0

Step 4: Processing message:

The fundamental procedure of the hash algorithm is the module which has four loops of preparing, each of 20 steps [17]. The contributions of each loop are 512-bits message chunk being overseen and the 160-bits buffer value (ABCDE). As the procedure proceeds with the estimations of the buffer are refreshed, the huge loop has a comparable development, but each stage with a different logical operation function which is parleyed as P, Q, R and S, which appeared in table 2.2. The yield of the last loop is summing of the contribution to the main loop in a strategy these bits of the yield are added to the reliable bits of the input [33] [34]. The processing appears in Figure 2.17.

Table 2.2: Primitive logic functions used in SHA-1

| step | Primitive logic function~ | $\sim(t.B.C.D)$ |
|-----------------------|---------------------------|--|
| $(0 \leq t \leq 19)$ | $P(t.B.C.D)$ | $\overline{(B \wedge C) \vee (\bar{B} \wedge D)}$ |
| $(20 \leq t \leq 39)$ | $Q(t.B.C.D)$ | $B \oplus C \oplus D$ |
| $(40 \leq t \leq 59)$ | $R(t.B.C.D)$ | $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ |
| $(60 \leq t \leq 79)$ | $S(t.B.C.D)$ | $B \oplus C \oplus D$ |

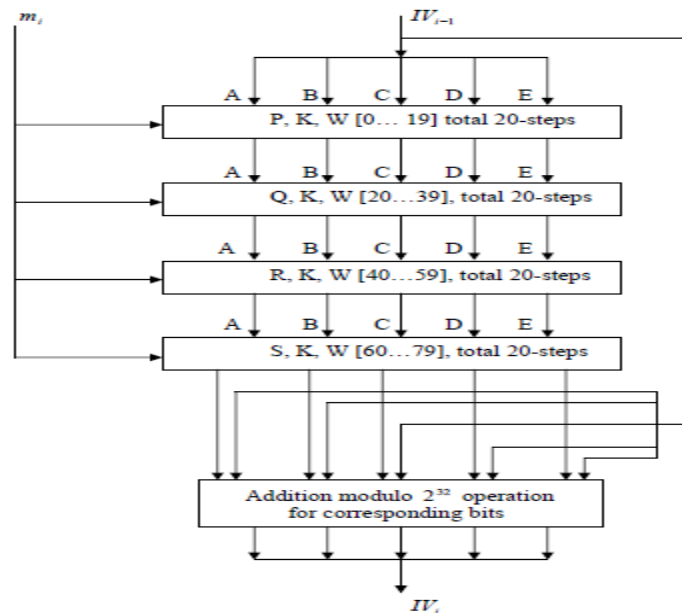


Figure 2.17: SHA-1 Compression Function [33]

Step 5: The Output:

Finally, the next procedure step states that all message blocks are handled in the previous way, the output of the final stage is a 160-bits message digest.

2.5.2 SHA-2

SHA2, not often used for now, is the successor of SHA1 and gathered 4 kinds of hash functions: SHA224, SHA256, SHA384 and SHA512. It works the same way than SHA1 but is stronger and generate a longer hash [35]. National institute of standards and technology (NIST) developed the new version of SHA, which is the SHA-2. Many network systems are still using the SHA-1 because it still strong and can modulate it to get more secrecy. Other systems start using the updated version of hash function which is SHA-2. NIST advanced SHA-2, takes a greater output (256 or 512-bits instead of the 160-bits in SHA-1) and changes inside the message calculation [36]. SHA-2 has many types giving to the number of bits of the message digest. The SHA-2 256 is described as an example, the SHA-2 256 algorithm basically consists of three stages: (a) message padding parsing (b) expansion and (c) compression.

a. Message Padding

The input message converted to its binary equivalent in order to be handled. This will happen by adding a '1' and padding with zeros until its length $\equiv 448 \pmod{512}$ [37]. The original message length is then appended to the end of the padded message as a 64-bit binary number, such as SHA-1.

b. Message Parsing

The resulting padded message is partitioned into N 512-bit blocks, denoted $M(1), M(2), \dots, M(N)$. These $M(i)$ message blocks are processed distinctly to the message expander.

c. Message Expansion

The SHA-2 256 algorithm functions work on 32-bit words, so that each 512-bit $M(i)$ block from the padding stage is observed as sixteen 32-bit blocks signified $M(i)_t, 0 \leq t \leq 15$. The message expander takes each $M(i)$ and expands it into 64 32-bit W_t blocks [37].

d. Message Compression

The W_t words from message expansion stage are then passed to the SHA compression function or the SHA core. The core uses 8 working factors named A, B, \dots, H which are then instated to predefined values $H_0(0) - H_7(0)$ toward the begin of each call to the hash function [38].

2.6 ATTACKS ON THE PROPOSED SYSTEM

An attack is any attempt to pulverize, uncover, adjust, handicap, take or increase unauthorized access to or make unauthorized utilization of an advantage. This section contains several attacks that occur on the two different methods: Advanced Encryption Standard and hash function.

A. Brute Force Attack

In advanced encryption standard, brute force attack includes formally checking all potential key combinations until the proper key is found. It is one way attack when it is not potential to take a use of other weaknesses in an encryption system [39].

AES has never been broken yet and it safe against every brute force attacks contrary to confidence and excuses. However, the key size utilized for encryption should always be large enough that it could not be broken by modern computers.

In secure hash algorithm, Brute force attack is a particular arrangement used to attempting subjectively computed hashes to get a correct message process. Brute force attacks don't rely on upon the development of the hash function. The security of any hash algorithm dishonesties on the length of yield hash code. Along these lines, the more drawn out hash code the more protected hash work. This kind of assault relies on upon wonders of experimentation to get a coveted hash function. For instance, brute-force attack is a lexicon attack which cover a tilt of word reference file string words to attempt them all in a progressive way [40].

B. Known-key distinguisher

Known-key distinguisher was first known to analyze the security of AES-128 and some Feistel-based ciphers. The purpose of this model was to have a better estimation of the security margin of a cipher, but also to cover the scenario of block cipher-based hashing, where the key is recognized and even selected by the attacker. The property offered for their distinguisher was an integral structure on the input and output of a set of plaintext/ciphertext pairs, for a given known key [41].

C. Related-Key Attacks

1. The chosen key attack: In the chosen key attacks, two linked keys with certain relationship are utilized and many plaintexts are encrypted under each of them. The attacker distinguishes only the relationship between the two keys, but not the keys. Attacker receives the ciphertexts and uses them to find both keys [42]. There are two types of chosen key attack: a chosen key known as plaintext attack in which only the relation between the keys is selected by the attacker and a chosen key chosen plaintext attack in which the attacker selects the relation between the keys and the plaintexts to be ciphered. These attacks not depend on the exact number of rounds of the attacked cryptosystem and even if the number of rounds is increased (especially if doubled). The resulting cryptosystem stays vulnerable to the same attack

2. The Chosen plaintext attacks: This attack can be combined with the attacks based on complementation properties and it is almost three times faster than the corresponding attacks based only on complementation properties. When this attack is used against 64-bits block ciphers, it requires about 2^{32} - 2^{37} chosen plaintexts, whose corresponding ciphertexts are to be stored in random access memory during the analysis [42]. A hash function will be destroyed if an aggressor stays capable to acquire the usage of the hash work encroaches in any event single of the claimed security property. On the off chance that a hash algorithm function is asked for to be collision resistant, a positive attack is to identify one impact no less than two divergent messages have a similar hash code [5]. There are a great deal of sorts of hash function attacks, however the outlined is just attacks that are free of the algorithm which additionally have many sorts yet the usually utilized are: birthday attack, Random attack and brute force attack [33].

D. Birthday Attack

The idea of birthday attack is created from birthday paradox. The birthday paradox conditions are a group of 23 arbitrarily chosen people the probability of two people as a minimum having the same birthday, is in excess of $\frac{1}{2}$ [32]. This called birthday attack which is utilized to deliver a well-known cryptographic attack. To property this, by assuming that the hash of the message of length n bits that gives 2^n potential outcomes to the hash code. "In the event that two pools from the process space, one containing $1 \times$ tests and the other covering $2 \times$ tests are made by a cryptanalyst, the likelihood of finding a match between two pools approximated given as" [31][33]:

$$p \approx 1 - 1/\left(e^{-\frac{x_1 \times x_2}{2^n}}\right) \quad (2.9)$$

In the theory of probability, the birthday issue identifies with the probability that in an arrangement of n subjectively chose individuals any match of them have a similar birthday. Not at all like the ordinary likelihood, the required number n of individuals that brand the likelihood of about combining having the common birthday more noteworthy than 0.5 is not firmly 180, it is just 23. For 57 individuals, the likelihood of finding any combine having shared birthday is over 99%. In genuine condition, the essential point of the aggressor is the fake of advanced marks in messages that the genuine sender does not have any desire to send. The same messages

that shift in just a couple of bits, for instance a lowercase letter supplanted by a similar letter, however a capitalized, there is a primary change in linking advanced marks. To defeat this drop, the attacker produces two arrangements of conceivable messages M_1 and M_2 . The first incorporates messages got from M_1 that the sender would be anxious to check, and that are evidently the same, yet fluctuate in a couple of bits. The second one incorporates messages, got from M_2 by changing a couple of bits, and is all messages that the aggressor needs to send. The soul of this strategy is to discover reasonable sets $M'_1 \in M_1$ and $M'_2 \in M_2$ so that:

$$h(M'_1) = h(M'_2) \quad (2.10)$$

Collision resistance is essentially noteworthy for digital signature robbery discouragement. At that point, if a collision between at least two messages happens, certain message's digital signature sent by sender contorted and included against a discretionarily picked message without that sender's endorsement or learning. The time craved to discover an impact is the vital occasions in evaluating a hash algorithm. This time is commonly called the pursuit cost of the algorithm [34].

E. Random Attack

The attacker chooses a message or bits of a message and prospects that is indistinguishable to the genuine message. On the off chance that the hash work expends the required self-assertive execution, then the likelihood of achievement is equivalent to $1/2^a$, where, a : is the aggregate number of bits of the hash code. For a Message Confirmation Code (MAC) the attack relies on upon two elements [33]:

1. The quantity of endeavors.
2. The normal incentive for a fruitful attack

2.7 FIELD PROGRAMMABLE GATE ARRAY (FPGA)

Field Programmable Gate Array is a programmable integrated circuit that includes a huge digits of logic cells with programmable correlation between them [43]. FPGA consists of thousands of universal building blocks, known as configurable logic blocks (CLBs) [3], connected using

programmable interconnects. Figure 2.18 shows how the FPGA made up of a Configurable Logic Blocks (CLB) that is linked by interconnections and input/output blocks [44]. Every CLB can have a high degree of a logic function for a supposed inputs. A VHDL is a hardware language which can be utilized to show the FPGA hardware functions. VHDL code converted to a binary bits stream when it written, and loaded into the FPGA system.

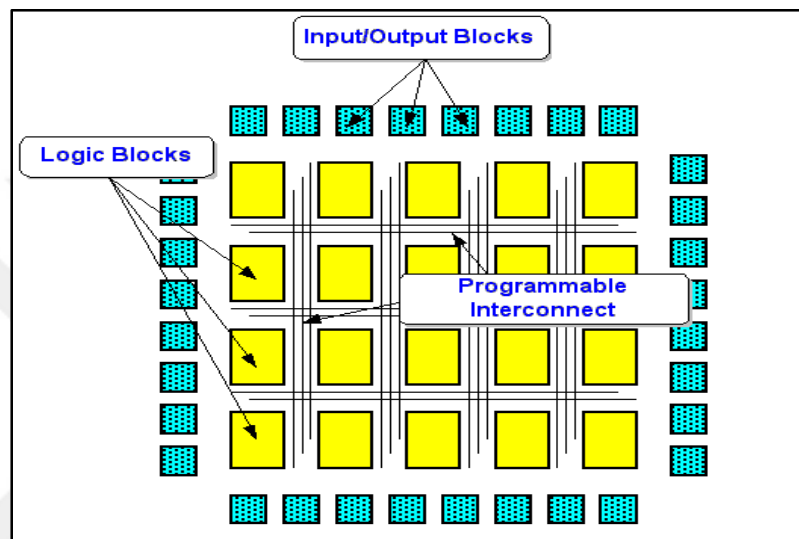


Figure 2.18: Main Structure of the FPGA [45]

FPGAs allow creators to alter their plans late in the configuration cycle-even if the program has been invented and sent to the field. FPGAs are a total fit for different markets because of the programmable nature of it. Xilinx is one of the biggest providers of FPGA tools. These tools are appropriate to be performed in a lot of applications like security, wireless communications, digital signal processing and other. To translate the design into a hardware particular software is needed. Software programs that was used in this work are; Labview 2013, MATLAB 2012a, ISE 14.7 and Xilinx System Generator 14.7. System Generator is a Xilinx design instrument that found in MATLAB based on block sets of Simulink. Steps of execution that done to create the FPGA program files are performed automatically by the Xilinx System Generator. System generator offers the ability to simulate hardware scheme that is handled by the MATLAB software. System Generator has a block named black box that allows the VHDL code to be got into the MATLAB-Simulink and co-simulated by Xilinx ISE Simulator.

To prove the designed system on FPGA platform, the Hardware (HW) Co-Simulation must be performed after establishing the system by timing analysis and simulation. By the HW Co-Simulation, hardware simulation is examined by making a comparison between the MATLAB simulation results and the device simulation results due to the Xilinx System Generator supply bits-true and cycle-true outcomes. The Co-Simulation generates the bits stream file for the Xilinx System Generator designed blocks depending on the Xilinx ISE and the core of the System Generator. The bits stream would be filled into FPGA board through basic connection such as Ethernet or Joint Test Action Group (JTAG) in order to implement the hardware implementation by linking the PC to the Xilinx board in a locked loop.

2.8 ENCRYPTION QUALITY MEASURING TECHNIQUE

Many of the system's statistical analysis are presented and discussed in this section. These analyses are classified into a group of tests. These tests are used to check the system randomness according the key from the proposed system. These tests are: monobits test (frequency test), serial test, poker test, run test and autocorrelation test.

These tests check whether the system produced data are random or constant. Before performing any statistical test, it is necessary to transform the set of data to a 20,000 continuous bits, then apply every test separately [46].

A. Frequency Test (Monobits Test) :

This test consider the first test of randomness, it is used to determine the number of zeros or ones in the 20,000 bits. The system passes this test if the number of one's (or zeros) is between 9,654 and 10,346 [46].

Let n_0 , n_1 refers to the number of 0's and 1's in s , respectively while n is the total number of bits. The statistic used is:

$$X_1 = \frac{(n_0 - n_1)^2}{n} \quad (2.11)$$

B. Serial Test (Two bits Test)

Serial test is used to calculate if the number of occurrences of 00, 01, 10, and 11 as subsequences of (s) are nearly the same, as would be thought for a random sequence. Let n_0, n_1 refers to the number of 0's and 1's ins, respectively, and let $n_{00}, n_{01}, n_{10}, n_{11}$ refers to the number of occurrences of 00, 01, 10, 11 in s respectively. As noted $n_{00} + n_{01} + n_{10} + n_{11} = (n - 1)$, because the subsequences are allowed to overlap. The statistic equation used is:

$$x^2 = \frac{4}{n-1} + (n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n}(n_0^2 n_1^2) + 1 \quad (2.12)$$

Which almost follows a χ^2 distribution with 2 degrees of freedom if $n \geq 21$. Note: if block (000) is a (00) and (00) [47].

C. Poker test

Performing this test requires the continuous 20,000 bits to be divided into 5000 segment and 4 bits of every segment. Then, determining the number of occurrence of each 16 possibilities for the 5000 segments. So, if $x(i)$ is the achieved number of counting each 4 bits possibility of i , where i between 0 and 15, then

$$P = \frac{16}{5000} \sum_{i=0}^{15} (x(i))^2 - 5000 \quad (2.13)$$

If P is between 1.03 and 57.4, then the system passes this test [46].

D. Runs test

This test is used to count the number of sequential ones and zeros, in the 20,000 continuous bits stream of the data set. All run length should be counted and saved for all lengths which are ≥ 1 . The run length boundaries for system's passing are shown in Table 2.3.

The system passes the run test if all 12 intervals (6 for zeros and 6 for ones) occur in the required interval [46].

$$X_4 = \sum_{i=1}^K \frac{(b_i + e_i)^2}{e_i} + \sum_{i=1}^K \frac{(g_i - e_i)^2}{e_i} \quad (2.14)$$

Which almost follows a 2 distribution with $2k - 2$ degrees of freedom Note: (K) depend on the higher (i).

Table 2.3: The run length boundaries

| Length of run | Required interval |
|---------------|-------------------|
| 1 | 2,267 - 2,733 |
| 2 | 1,079 - 1,421 |
| 3 | 502 - 748 |
| 4 | 223 - 402 |
| 5 | 90 - 223 |
| 6 or greater | 90 - 223 |

E. Autocorrelation test

The aim of this test is to check for correlations between the sequence (s) and its shifted versions. Let d be a fixed integer, $1 \leq d \leq \lfloor n/2 \rfloor$, the number of bits in s not equal to their d-shifts is

$$A(d) = \sum_{i=0}^{n-d-1} S_i \wedge S_{i+d}, \text{ where } \wedge \text{ refers to the XOR operator.}$$

The statistic:

$$X5 = 2(A(d) - \left(n - \frac{d}{2}\right)) \sqrt{n-d} \quad (2.15)$$

Which almost keep track of an $N(0, 1)$ allocation if $n - d \geq 10$. For small values of A (d) which are not as expected as huge values of A (d), a two sided test should be utilized [47].

3. SOFTWARE IMPLEMENTATION OF AES AND HASH FUNCTIONS

3.1 INTRODUCTION

In this Chapter, AES and hash functions were implemented based on software process using LabView to give an overview of using AES, SHA-1, and SHA-2. Furthermore coupling SHA-1 and SHA-2 with AES to show the use of hash functions to achieve integrity and authentication by merging it with AES to achieve confidentiality and a two way cryptographic algorithm. AES and hash functions are also can be implemented as a hardware to construct a system of encryption and decryption by using FPGA.

3.2 SOFTWARE IMPLEMENTATION

In the first step of implementing, the AES and hash function algorithms (SHA-1, SHA-2) over FPGA was presented using National Instrument LabView 2013 version (32 bits). This presents an overview of cryptographic AES algorithm and hash function on LabView to work with logical calculations easily through the necessary functions of SHA-1 and SHA-2 that are designed in LabView.

Hash function achieves integrity and authentication, this made hash function itself does not achieve confidentiality. To make the hash more secure, there is a need to merge it with any encryption method. The AES algorithm was implemented in LabView to achieve the confidentiality property to support the security of the system. The original message that processed by the hash function algorithm first represent the plain text, while the hash code that produced via a hash function represent the key of the AES algorithm to generate another code which is very difficult to break.

The Advanced Encryption Standard software implementation was implemented by LabView as in the sections below:

3.2.1 AES Implementation by LabView

This section presents the implementation of the Advanced Encryption Standard (AES) algorithm using LabView. The steps of the proposed algorithm of AES which were implemented are: entering a plain text (string), AES encryption block, AES decryption block and finally the output plain text which is the same input plaintext that was first encrypted. The procedure of the AES implementation was based on LabView environment in order to be suitable for effective realization and implementation of cryptographic algorithms. Furthermore, if the AES (256 bits) will be implemented, the key length need to be changed. This will enable of using SHA-2 instead of SHA-1. The implementation steps of the AES software are as follows.

3.2.2 AES Encryption process by LabView

Before the message enters the case structure of the LabView AES block diagram, some definitions must be introduced. Firstly, create the variables of AES VI of LabView and the dimensions of AES as an array string and defining the key value so that to introduce the encrypt algorithm string as an array of bytes. If the length of the plaintext is less or equal to zero, then no plaintext is entered. The plaintext value is then entered as a dimension string and the AES entered as a bytes value. The value of the AES key is entered as (128, 192 or 256 bits), since that we would connect the hash function with AES where the hash function (SHA-1,SHA-2) is entered as a key (section 3.3 in chapter three shows the software implementation of hash functions SHA-1 and SHA-2 in) to the AES block diagram instead of entering the key directly, so the key used for the AES is 192 bits or 256 bits as the hash functions (SHA-1 and SHA-2) lengths are 160 bits and 256 bits respectively. The decryption algorithm is also introduced by decrypting the bytes into a string which can perform the stream transform. The stream contains all data such as the plaintext and the array bytes encrypted. The streams that created by the decryption algorithms are used for the encryption algorithm in order to return the encrypted bytess from the memory stream. This process is made to create an AES object with specified key and Initial Vectors (IV) and additionally, creating the encryption, decryption algorithm so it can be able to return the original plaintext.

The key value is chosen according to the length of the hash functions (SHA-1). For "SHA-1", 192 bits key length is used of the AES where SHA-1 will use 160 bits of the AES length and the

rest of the length is used as a salt. The cipher block chaining (CBC) mode is used. Cipher Block Chaining (CBC) is discussed in details in chapter two.

Each key value has its own number of rounds where AES-128, AES-192 and AES-256 has 10, 12 and 14 rounds respectively as discussed in chapter two. The more rounds used means more security against cryptanalysis. Simply, this means there is more confusion and diffusion.

The initial phase of the cipher is the key creation that happens on the server. In this stage, the client would give a confidential key that will be extended through utilizing the Rijndael key program. The small key is augmented to a bigger one. A key of 128-bits is changed into a key of 176-bytess, a key of 192-bitss is changed into a key of 208-bytes, and a key of 256-bitss is changed into a key of 240-bytess. The key schedule would take a 4-bytes part of the key as a digit 32 bytes and a repetition count and deliver this information to the key schedule core that restores a 256-bitss (32 bytes). This step can be seen in Figure 3.1.

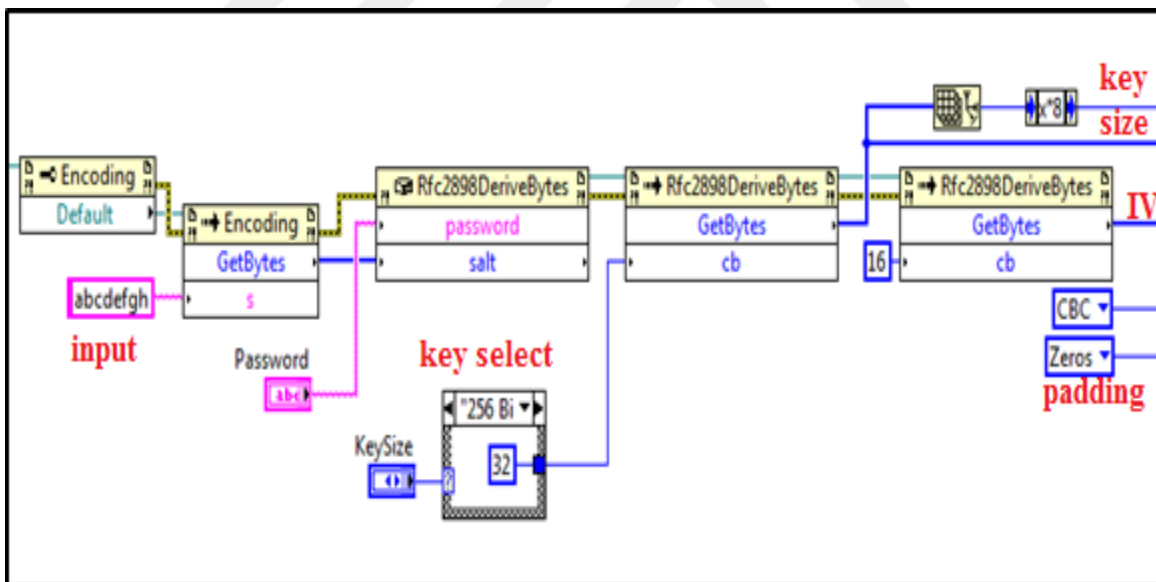


Figure 3.1: Key Generation in LabView

The key schedule core initially plays out a bytes shifting on the key and divides the 32 bytes into particular bytes and sends the bytes to have Rijndael S-box used. This is utilized to apart the connection between the key and the ciphertext.

The Rcon step is the next step in the core scheduler. This picks the principal bytes of the resulting word from the Sbox and plays out a XORing operation on the bytes with the outcomes of the Rcon step, which is basically the exponentiation of 2 to the repetition count of the key schedule.

After producing the key, the type of encryption to perform, data about the key and also the key itself should be transmitted to the scheduler. The key is changed into a bytes array and transformed one bytes at a period alongside a break to guarantee the schedule gets every part of the information in the right order.

The server then would send essential commands to the scheduler to progress the information by both the encryption and decryption process before restoring the information returned to the client.

If the server wants to have the capacity to play out the cryptanalysis, it is perfect to keep the key in a LUT (Look up Table). So a VI methodology is utilized. There are various cryptographic algorithms which the look up table can be utilized to store a wide range of keys which will give a client adaptability.

The moment that the key data is given from the server, the encryption could occur when the host hands the signal. The encryption would initially produce what is known as a 'round key' which is gotten from the key schedule. It would then start the relating procedure playing out real encryption.

The main repetition of the encryption algorithm will join every bytes of the state with the round key utilizing a bitwise XOR. The size of the repetition (named rounds) will play out a non-linear substitution that will substitute every bytes with other bytes according to a predefined look up table. At that point every row of the state will be shifted in a cyclic way.

After that, there would be a mixing operation that happens on the columns and combine four bytes of every column. Afterward the round key would be included once more. The last repetition is like the past "n" rounds except that the column mixing operation not included.

Key Expansion: key expansion routine is used to deliver the round keys from the cipher key. The AES standard specifies the key expansion operations on a four bytes words. The subkey is poised of four such words. In AES-256, the key expansion nearly univocal:

- The main subkey corresponds to the cipher key itself.
- The accompanying words are calculated recursively from this initial set of words utilizing a basic XOR function.
- For the words with files that are a multiple, a special transformation is used. At the beginning, the bytes requesting of to start with is changed by the cyclic left shift, and after that, the SubBytes function is connected to each of the four bytes. In the AES standard these operations are named RotWord and SubWord individually.

The AES encryption process is shown in Figure 3.2:

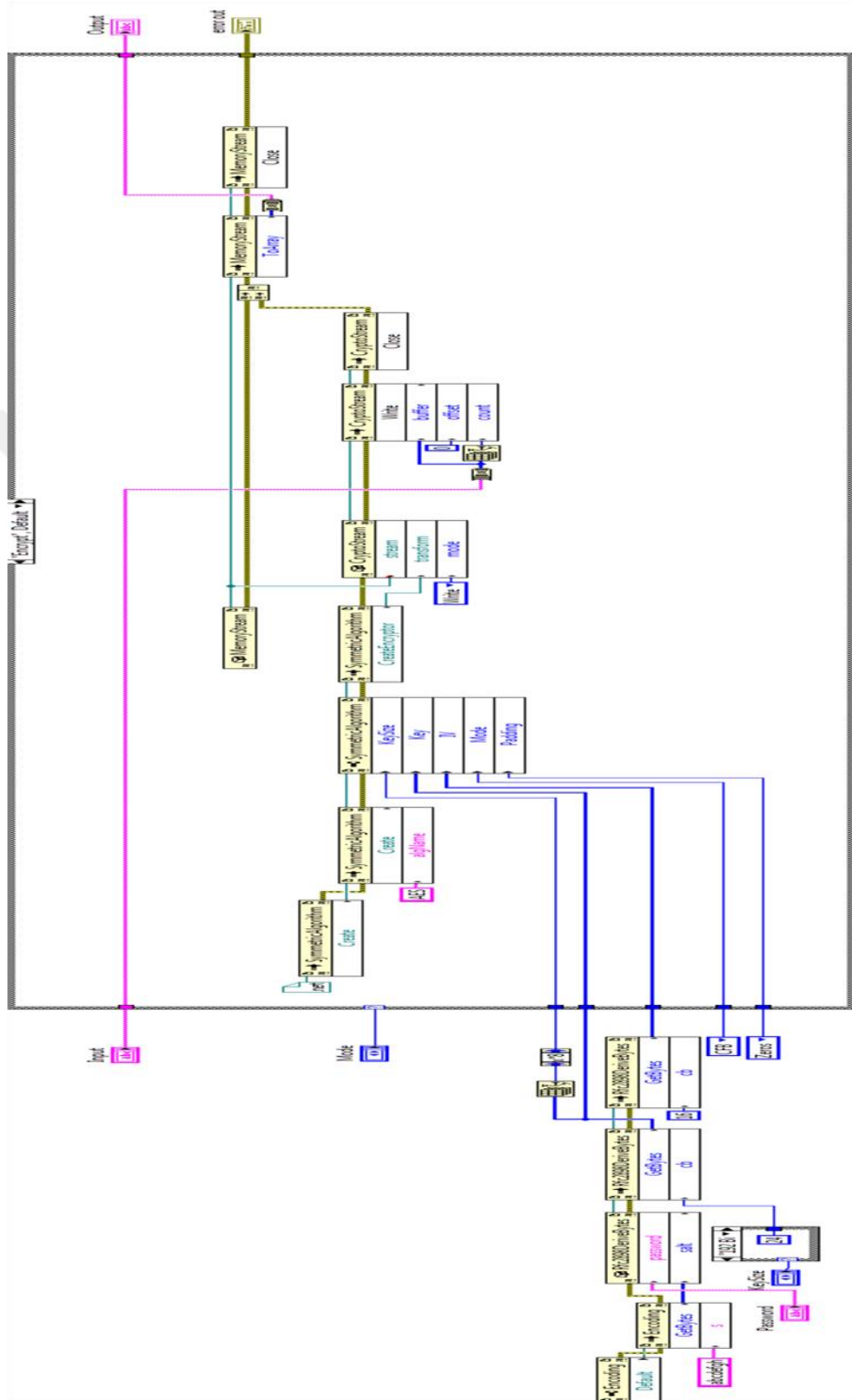


Figure 3.2: AES Encryption Process by Labview.

3.2.3 AES Decryption Process by LabView

Decrypting means doing everything in reverse. The same definitions that were introduced in the previous section that are used with the decryption process by LabView.

The algorithm was built via LabView, the initial values that introduced to encode in order to get bites and the string is also entered which is considered as the plain text input. The string and Bytes that entered are also encoded to get the salt that is added to complete the space left from the entered key. The key entered is the secure hash algorithm (section 3.3), which is 160 bits in length, therefore, to be comparable with the AES key size, 192 bits key length of the AES is used.

After entering the plaintext and the password is now produced by the hash algorithm, the key value is specified according to the length of the hash function (SHA-1) as we mentioned previously. The CBC mode is used.

The first step of the decryption algorithm is the inverse add round key. Inverse add round key transformation is comparable to the forward add round key transformation, since the XOR has its own inverse operation. The next step of the AES decryption is the inverse mix columns. The procedure of the inverse mix column is implemented in Equation (2.5) in chapter two.

The next step of the AES decryption mode is the inverse shift rows. Inverse shift rows is the reverse of the shift row transformation. The bytes that are founded in the last three rows of the state are consistently transferred over different quantities of bytes. The first line is not moved. The last three rows are consistently transferred by $Nb - \text{shift}(r, Nb)$ bytes, where the move esteem $\text{shift}(r, Nb)$ depends on the line number.

The last step of the AES decryption mode is the inverse sub bytes. This step is the reverse of the bytes substitution transformation, in which the inverse Sbox is applied to each bytes of the State. This is obtained by applying the inverse of the transformation followed by taking the multiplicative inverse in GF (28). The inverse S-box used in this step transformation is presented in Table (A.2) in Appendix A.

Finally the output is converted from an array to a string and returns the plain text entered in the first place. The result of this operation can be seen in details in chapter four, different modes are used like OFB mode, CTR mode and else.

The AES decryption process is shown in Figure 3.3.



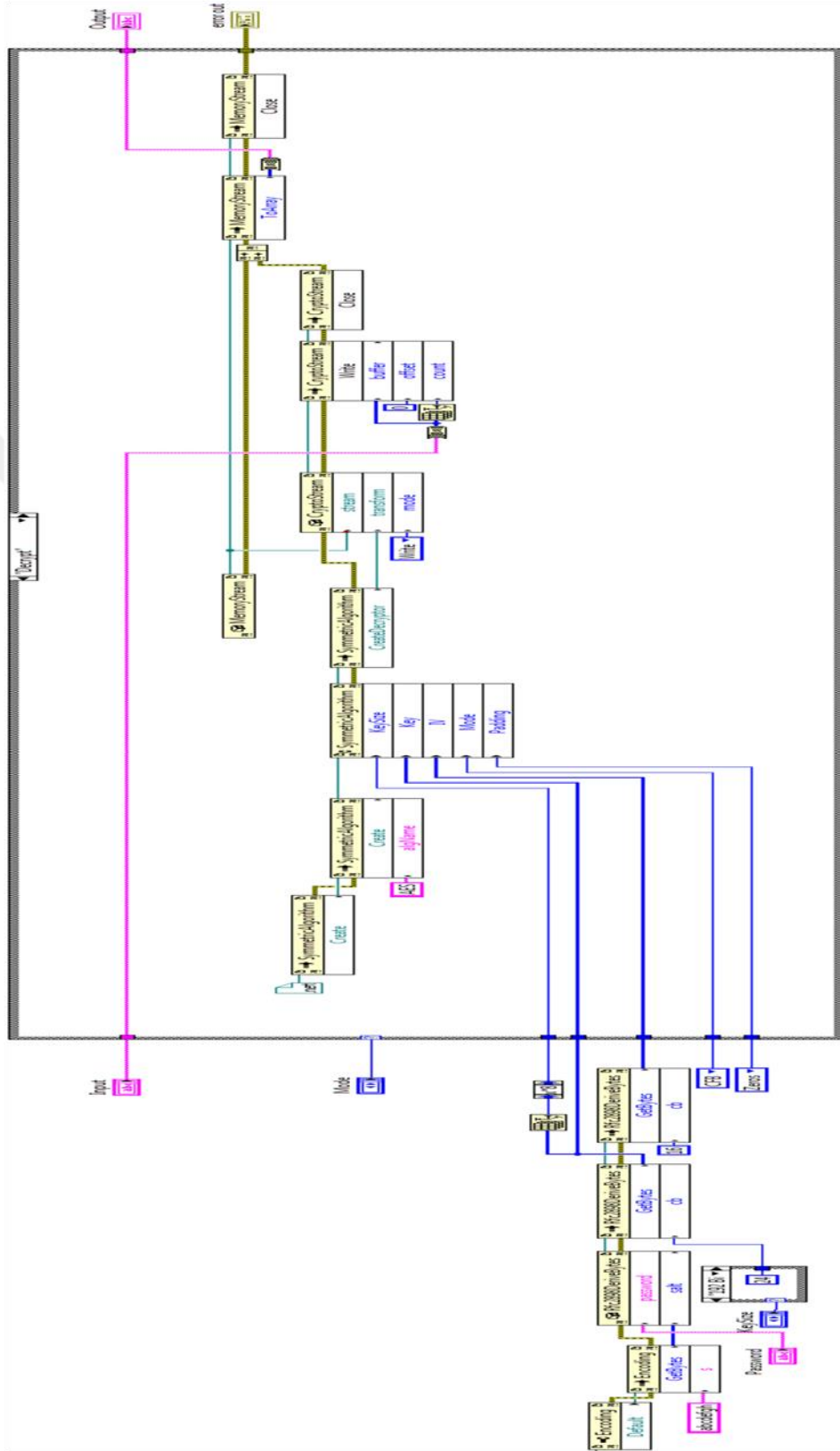


Figure 3.3: AES Decryption Process by Labview

3.3 THE IMPLEMENTATION OF HASH FUNCTIONS BASED ON LABVIEW

The software implementation of the hash functions (SHA-1 and SHA-2) is implemented as it will describe in the next sections.

3.3.1 The Implementation of SHA-1

Secure Hash algorithm (SHA-1) is an extremely significant calculation for integrity and authentication acknowledgment. The proposed calculation of SHA-1 actualized by LabView programming from entering string, padding, SHA-1 core, and message digest to deliver 160 bits hash code for any plaintext.

The principal organize at the padding procedure is the message entering to the framework chunk by chunk which implies each 512 bits are handled alone with succession However, enters as a bytes (64 bytes) that is particular at Figure (3.4). The symbols in LabView appeared in appendix A.

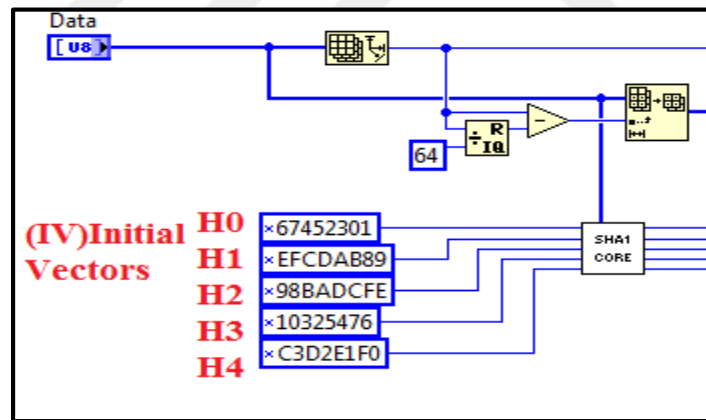


Figure 3.4: Initial Vectors of SHA-1

The message is entered to the SHA-1 center. The initial vector's values as observed in Figure (3.4) (H0, H1, H2, H3, H4) are instated as settled constants, for instance, H0=67452301, H1=FEC DAB89, H2= 98BADCFE, H3=10325476, and H4=C3D2E1F0. The message separated into chunks of size 512.

The information that was prepared by the SHA-1 core, is the padded message (or the message with 512 bits measure without padding) and the initial vectors to be handled with their initial

worth and after that refreshed after every figuring inside the center. The initial vectors qualities were in Hex, so ought to be changed over to their proportional values in unsigned whole number then to their ASCII to be good with the message array. The message measure entered to the center is with the extent of 512 bits, yet the procedure work with information bytes by bytes, so its 64 bytes that is the most extreme size of chunk to be prepared at once.

The entire computations included inside for loop in LabView program which execute the operation an N of times.

The principal sub-step in SHA-1 core which is that each chunk would be putted through a small function that would make 80 words from the 16 words that was entered. This progression is a loop t means each progression after that would be rehashed till a specific condition is valid. This situation would start by tuning the variable "i" equivalent to 16. After every run of the loop, a 1 would added to "i" until "i" is equal to 79.

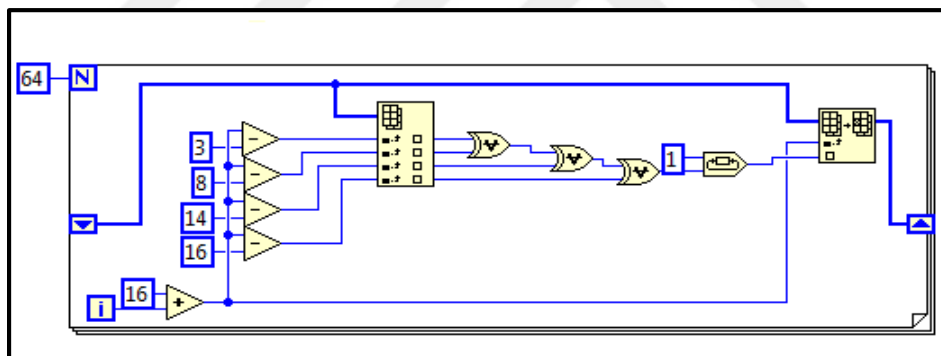


Figure 3.5: SHA-1 core word calculation

In the wake of apportioning the 80 words array the technique comes progressively, saves the initial 16 words to be the 512 bits block are isolated into 16 words, the rest of the words are made by playing out the logical function: Word [i3] XOR word [i8] XOR word [i14] XOR word [i16], and afterward rotated left as a one bits, as appeared in Figure 3.5. The LabView block diagram demonstrates that the third, eighth, fourteenth, and sixteenth words XORed and rotated to left.

Next, the functions that represent the calculations and the constants identified a sequence of logical functions of the hash core process are: $f(0), f(1) \dots f(79)$ is utilized in SHA-1. Every $f(t)$,

$0 \leq t \leq 79$, works on three 32-bits words B, C, D and achieves a 32-bits word as output. $f(t;B,C,D)$ is introduced as follows:

$$f(t;B,C,D) = (BANDC)OR((NOTB)ANDD) \quad (0 \leq t \leq 19) \quad (3.1)$$

$$f(t;B,C,D) = BXORCXORD \quad (20 \leq t \leq 39) \quad (3.2)$$

$$f(t;B,C,D) = (BANDC)OR(BANDD)OR(CANDD) \quad (40 \leq t \leq 59) \quad (3.3)$$

$$f(t;B,C,D) = BXORCXORD \quad (60 \leq t \leq 79) \quad (3.4)$$

These operations implemented in Labview blocks as shown in Figure 3.6:

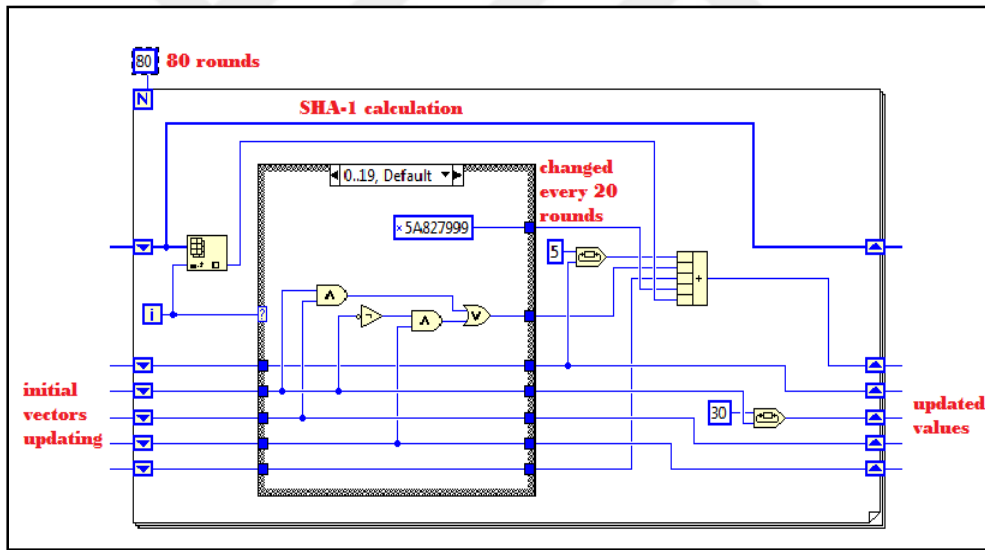


Figure 3.6: SHA-1 calculations

A series of constant words $K(0), K(1) \dots K(79)$ is used in the SHA-1. In hex these values are as the following:

$$\left. \begin{aligned} K(t) &= 5A827999 & (0 \leq t \leq 19) \\ K(t) &= 6ED9EBA1 & (20 \leq t \leq 39) \\ K(t) &= 8F1BBCDC & (40 \leq t \leq 59) \\ K(t) &= CA62C1D6 & (60 \leq t \leq 79) \end{aligned} \right\} \quad (3.5)$$

From Figure 3.6, the 80 rounds of hash algorithm started. Inside these rounds, there is a case structure which has four iterations, for each iteration the logical operation converted and also the constant changed according to t value as seen in Figure 3.7.

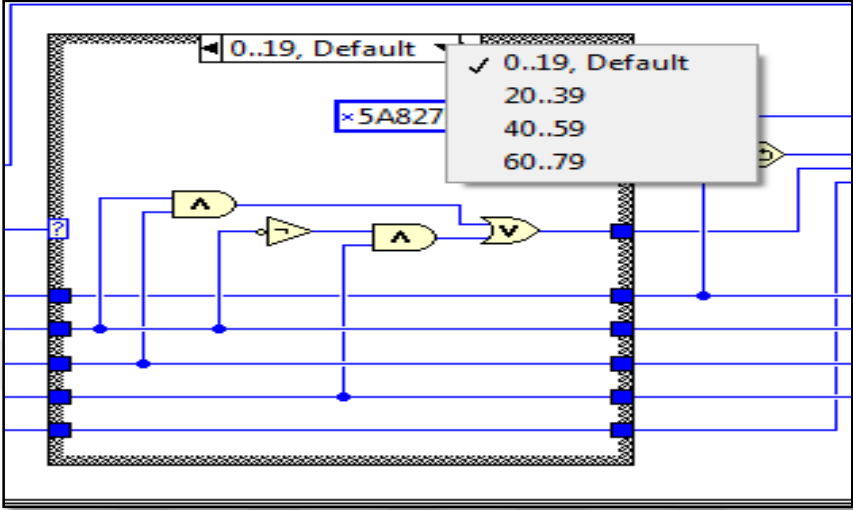


Figure 3. 7: iterations of the case structure

The message process is spoken to in Hex to give 160 bits hash code of the SHA-1 calculation. The last phase of the SHA-1 core is the message process which is the last computation to give the last code which is the hash.

The whole algorithm of SHA-1 implemented by LabView is shown at Figure 3.8.

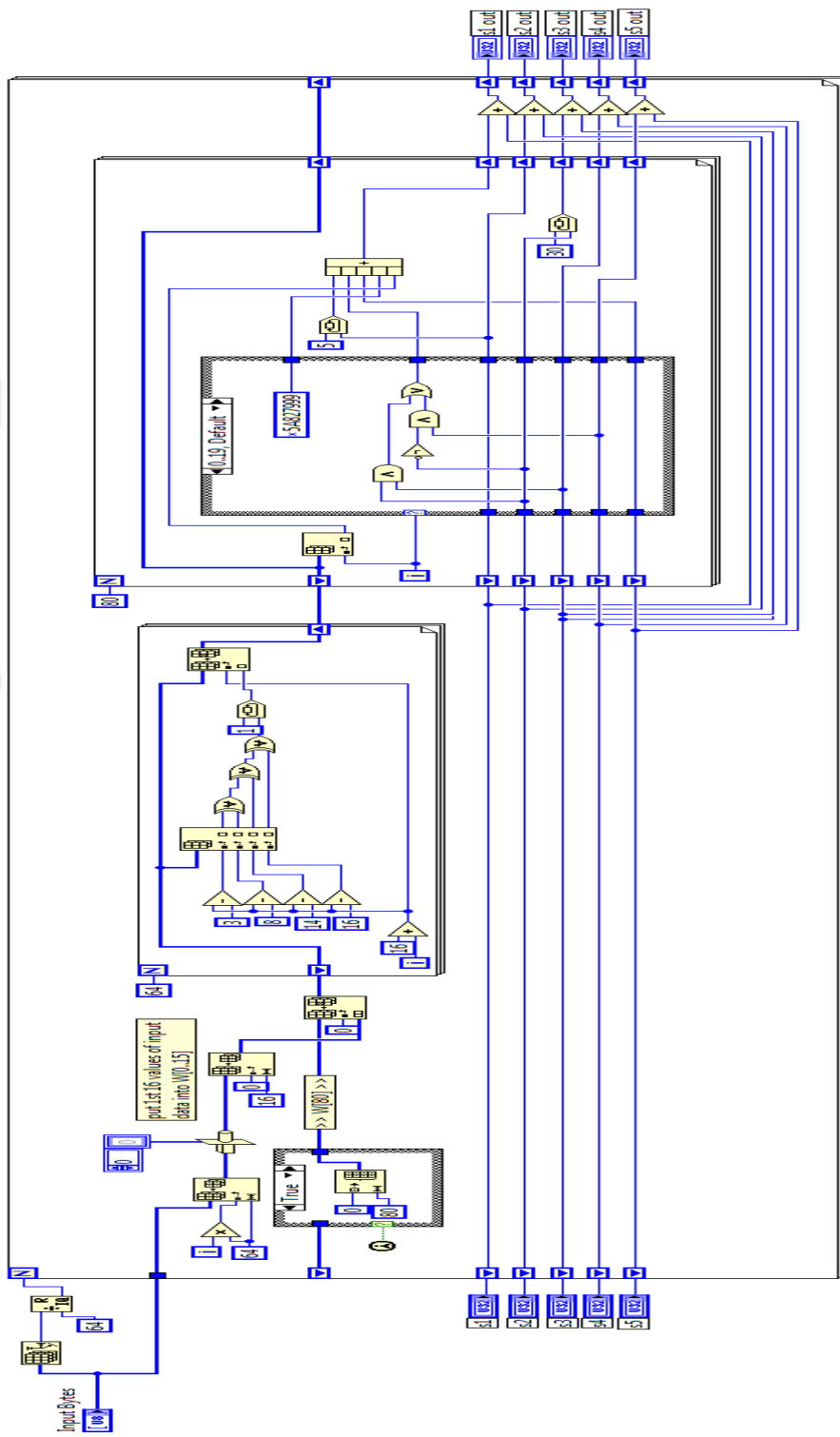


Figure 3.8: SHA-1 by Labview

3.3.2 The Implementation of SHA-2

The implementation of secure hash algorithm (SHA-2 256) is performed in this section. The algorithm of SHA-2 was implemented, from entering string, padding, SHA-2 core, and message digest to achieve 256 bits hash code for any entered plaintext.

Calculation of a hashed message starts with preparing the message. Pad the message in the usual way (as in SHA-1), let (M) be the message and (L) be the length of the message in bits. Add bits "1" to the end of the message, and then (k) zero bits where (k) is the shortest non negative solution to the equation $L+1+k=448 \pmod{512}$. Add the 64-bits block that is equal to the number written in binary. As an example the 8-bits ASCII message "a b c" has length $8 \cdot 3 = 24$, so it is padded with a one, then $448-(24+1) = 423$ zero bits, and then its length to become the 512 bits padded message.

01100001 01100010 01100011 1 $\underbrace{00 \dots 0}_{423}$ $\underbrace{00 \dots 011000}_{64}$.

The entered length of the padded message should then be a multiple of 512 bits.

Next step is to analyze the message into a number of an (N) 512 bits blocks $M(1), M(2) \dots M(N)$. The first 32 bits of the message block (i) are denoted M_0^i , the next 32 bits are M_1^i , and so on up to M_{15}^i . With every 32 bits word, the left-most bits is saved in the most significant bits position as shown in Figure 3.9.

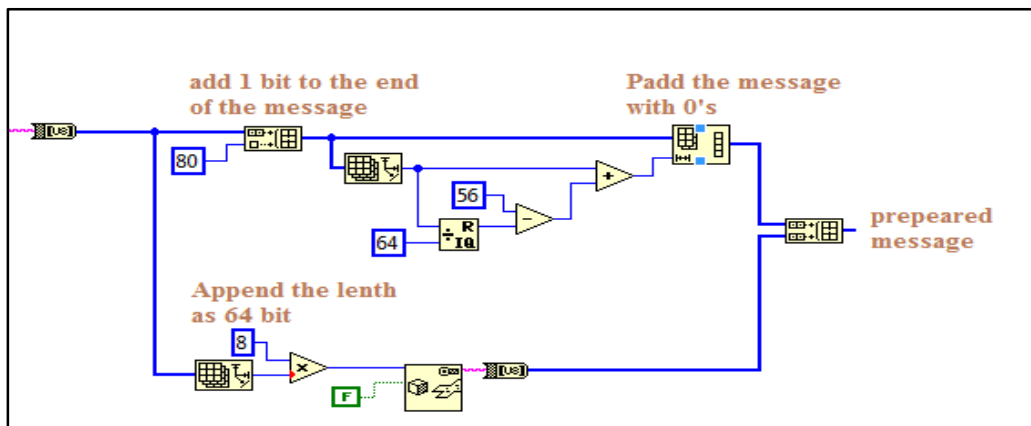


Figure 3.9: Message Preparation

At the end of the process, the prepared message is given as an array in ASCII code, just similar to SHA-1. After that, the array of (ASCII) should be converted to 32 word size to be appropriate for applying to the next step of the process of the SHA-2 core. This function is ready by NI to convert an array from U8 to U32, as shown in Figure 3.10.

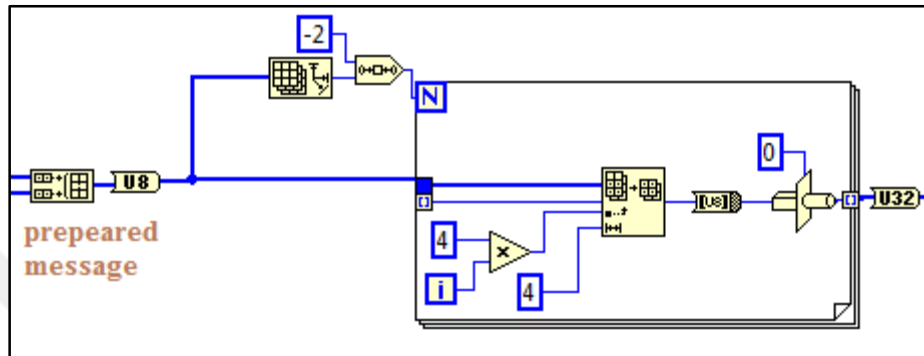


Figure 3.10: Converting Array

The converted array that introduced to the SHA-2 core simultaneously with the entry of the initial vectors.

The (SHA-2) Core holds the whole system calculations and functions. Initial vectors and padded message are the entry to the SHA-2 core.

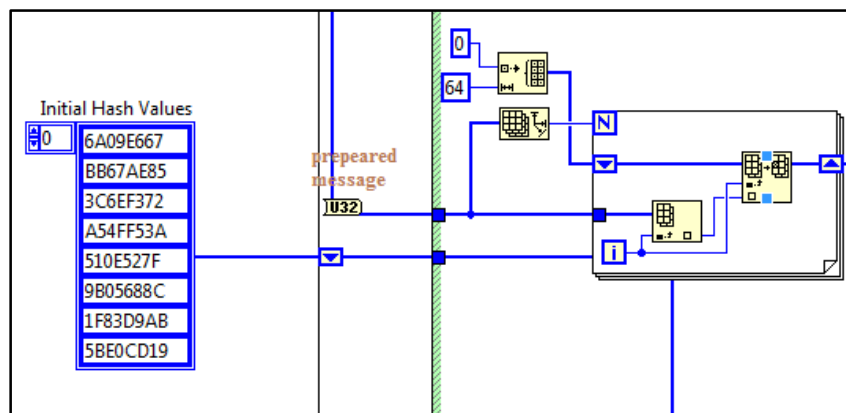


Figure 3.11: SHA-2 core Entry

The next step is to define a working array by the array initialization function. Like (SHA-1, SHA-2 256) treats message chunk by chunk, each with size of 512 bits, then the existence of a 64-entry message schedule array w [0...63] of 32-bits words is completed for each chunk. After

this process, copying chunk to the top 16 words $w[0..15]$ of the message schedule array is done and the initial hash values were entered to the system, as shown in Figure 3.10. Converting the initial 16 words into the residual 48 words $w[16..63]$ of the message schedule array is in Figure 3.12.

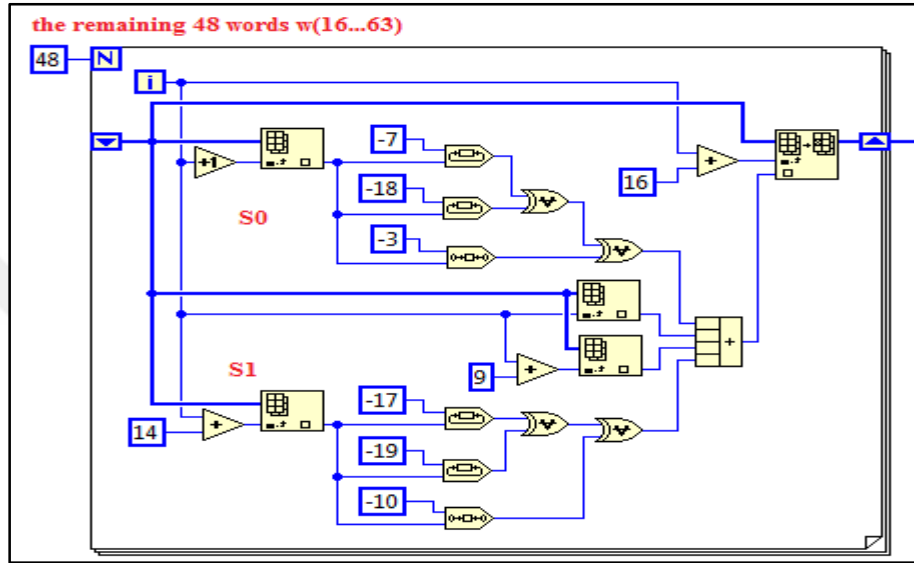


Figure 3.12: Working Array

After that, similar to SHA-1, the remaining 48 words pass through a different step from SHA-1. This different step is responsible for the logical operations, identifying two temporary variables: S_0 , S_1 and their values updated according to the following:

For i from 16 to 64

$$s_0 = (W[i - 15] \text{ ROT } 7) \text{ xor } (W[i - 15] \text{ ROT } 18) \text{ xor } (W[i - 15] \text{ SHR } 3) \quad (3.6)$$

$$s_1 = (W[i - 2] \text{ ROT } 17) \text{ xor } (W[i - 2] \text{ ROT } 19) \text{ xor } (W[i - 2] \text{ SHR } 10) \quad (3.7)$$

$$W[i] = W[i - 16] + s_0 + W[i - 7] + s_1 \quad (3.8)$$

Next step is to define variables to present hash value: A= H0, B= H1, C= H2, D= H3, E= H4, F= H5, G= H6 and H= H7. These variables are defined to be assigned by the old values of the initial vectors in order to be updated at the next step of processing, as shown in Figure 3.13.

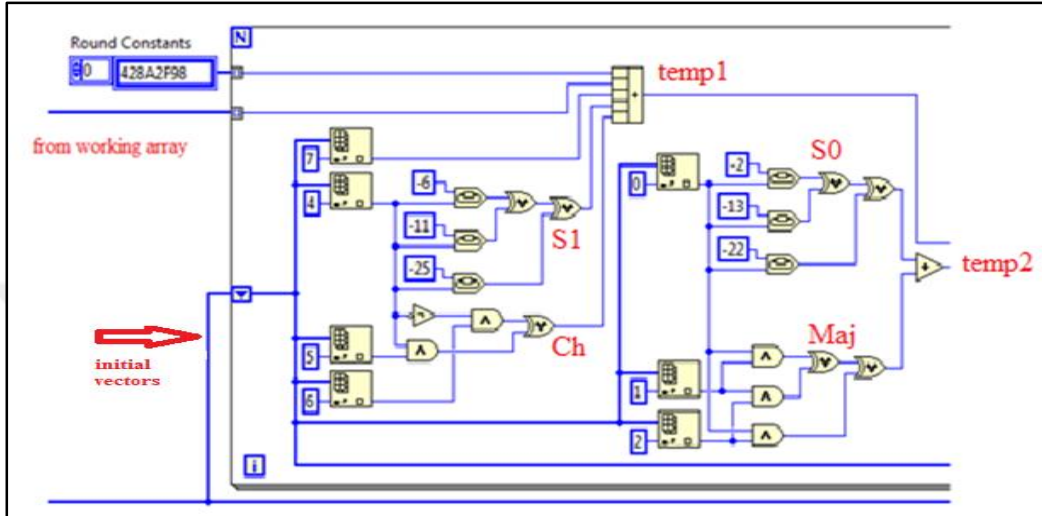


Figure 3.13: SHA-2 Calculations

After that, the compression function of the (SHA-2) main loop is initialized and the working array from Figure 3.13, and the constants of the round are stepped inside to the loop. The constant values are changed after every iteration which means; there are 64 constants for SHA-2 while SHA-1 has only four constants. Two temporary registers are defined, they are temp1, temp2. After that the values of the variables are updated as

$$\begin{aligned}
 H &= G, G = F, F = E, E = D + temp1, D = C, C = B, B = A, \\
 A &= temp1 + temp2
 \end{aligned}
 \tag{3.9}$$

Then the Compression hash operation main loop starts:

For i from 0 to 63

$$S1 := (EROT\ 6) \text{ xor } (EROT\ 11) \text{ xor } (EROT\ 25) \quad (3.10)$$

$$Ch := (EANDF) \text{ xor } ((NOTE) \text{ AND}G) \quad (3.11)$$

$$temp1 := H + S1 + Ch + k[i] + w[i] \quad (3.12)$$

$$S0 := (AROT\ 2) \text{ xor } (AROT\ 13) \text{ xor } (AROT\ 22) \quad (3.13)$$

$$Maj := (AANDB) \text{ xor } (AANDC) \text{ xor } (BANDC) \quad (3.14)$$

$$temp2 := S0 + Maj \quad (3.15)$$

Maj and Ch are symbols from FIPS standards which are still ambiguous to the meaning of what this symbols stand for.

The last step of SHA-2 algorithm is updating the values by adding the old values which are IV (the initial vectors) and the updating values resulted from Figure 3.13 to achieve the hash code and then it would be converted to its equivalent Hex value.

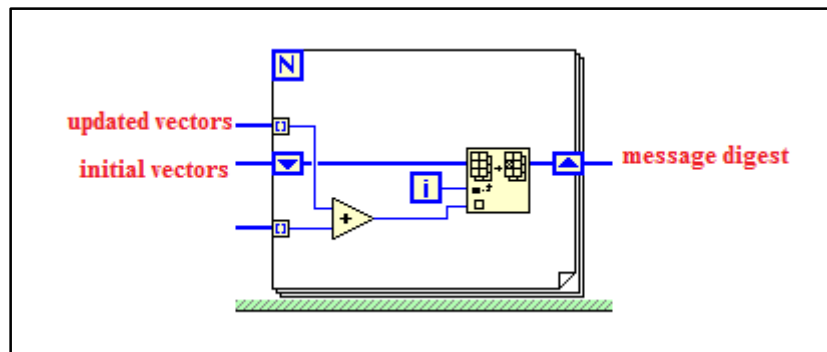


Figure 3.14: Message Digest

The full description of SHA-2 algorithm is illustrated in Figure 3.15.

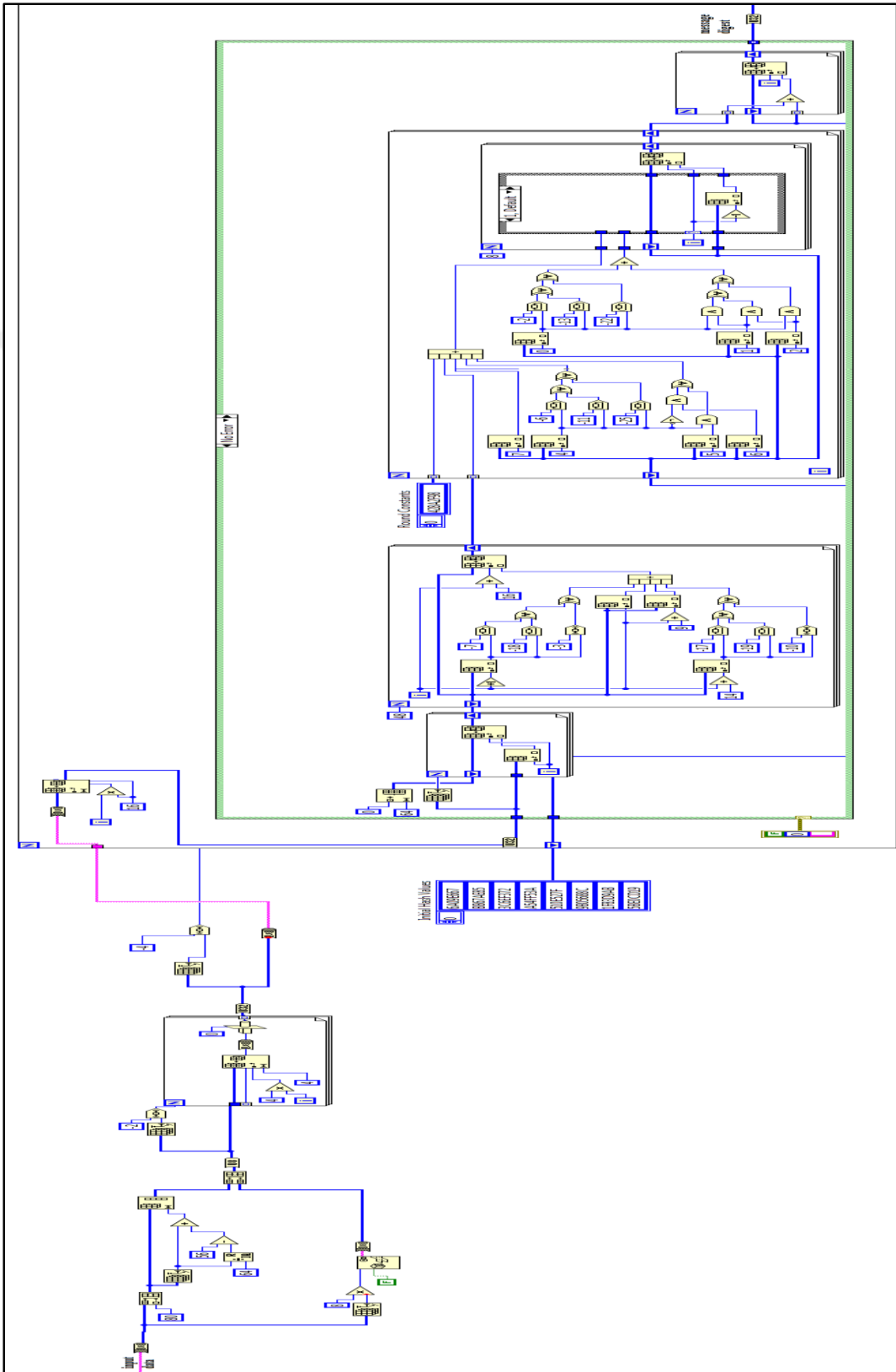


Figure 3.15: SHA-2 by Labview

3.4 COMPARATIVE ANALYSIS

A scientific examination for the correlation of regular and proposed calculation is displayed in this section. In this thesis, the size of key used in the hardware implementation is 128 bits, so the whole key space will be of 2^{128} Combinations which is equal to (340282366920938463463374607431 768211456).

Therefore; system that is able to perform combination for each unit of time would have (340282366920938463463374607431768211456) time, units to discover the key via an attacker. Furthermore, the size of the input plaintext is of 128 bits, all the input data space would be same as key space that is equal to (340282366920938463463374607431768211456) in addition to the same quantity of time units that needs to discover the data by the attackers. Furthermore, there are 2 attempts to find a transpose array.

Therefore, the total space will be consisting of 2^2 Combinations that equal 4. The Xor-ing will be consisting of 2^1 Combinations that equal 2 operations. Table 3.1 shows the result combination of brute force attack.

As illustrated in this table the maximum attempts to discover the cipher by force attack per second are (680564733841876926926749214863536422918) which is a very large number.

Table 3.1: Comparative analysis of the proposed encryption algorithm

| S | Reference type | Bits size(n) | Types Techniques(n) | No. of Attempt(2^n) | No. of Attempt/Sec (Complexity) |
|-----|------------------------|--------------|---------------------|-------------------------|---|
| 1 | Input Data | 128 | | 2^{128} | 340282366920938463463374607431768211456 |
| 2 | Input Key | 128 | | 2^{128} | 340282366920938463463374607431768211456 |
| 3 | Transpose Method | | 2 | 2^2 | 4 |
| 4 | Exclusive-OR operation | | 1 | 2^1 | 2 |
| Sum | | | | | 680564733841876926926749214863536422918 |

4. THE RESULTS OF SOFTWARE AND HARDWARE IMPLEMENTATION

4.1 INTRODUCTION

In this chapter, the results of the system design of encryption and decryption processes of the AES and SHA algorithm are presented. As mentioned previously, the simulation based on MATLAB (2012a) and Labview (2013) 64-bits. The proposed algorithm was written as a VHDL model by using the Xilinx ISE 14.7 (Integrated Software Environment) and input/output ports of an existing sample, this sample is found in block sets of Xilinx System Generator as function blocks that is found in MATLAB/Simulink. The origin files of the VHDL code created in ISE are brought to the System Generator by black box block and simulated by MATLAB/Simulink connected with Xilinx System Generator. The hardware program that was utilized to implement the proposed algorithms system was Spartan 3A 700AN.

4.2 AES AND HASH FUNCTIONS SOFTWARE RESULTS

This section describes and shows the result coupling AES with Hash Functions (SHA-1 and SHA-2) algorithms at each step of software implementation by LabView 2013 (32 bits).

A. AES and SHA-1 Software Result by LabView

From the implementation of cryptographic Advanced Encryption Standard algorithm and hash function in LabVIEW 2013 that described in chapter 3, the test of execution of all steps in AES and SHA-1 algorithms which held for different types of plain text such as English, Arabic, symbols, and numbers to produce fixed 128 bits AES code and 128 bits hash code. The testing Figures of AES and SHA-1 hash function are shown in Figures 4.1, 4.2 and 4.3, the cipher block chaining (CBC) mode is used.

The Salt is a strong random value that is added to the input of hash functions creating unique hashes for every input data, regardless of the input not being unique. Salt is a cryptographically constant length, it considered not determinism a hash function and AES, which is a great as long as duplication of password is something we don't want to being discovered.

The cipher text is a hashing code which converted to string to be inserted as a password to AES that will be combined with salt. Salt could be any character, number or symbol. Salt will be fixed to (q12) for example to see the difference between the Encryption Modes, it's could be variant due to the need such as daily Salt, hourly Salt or even been agreed between the transmitter and receiver.

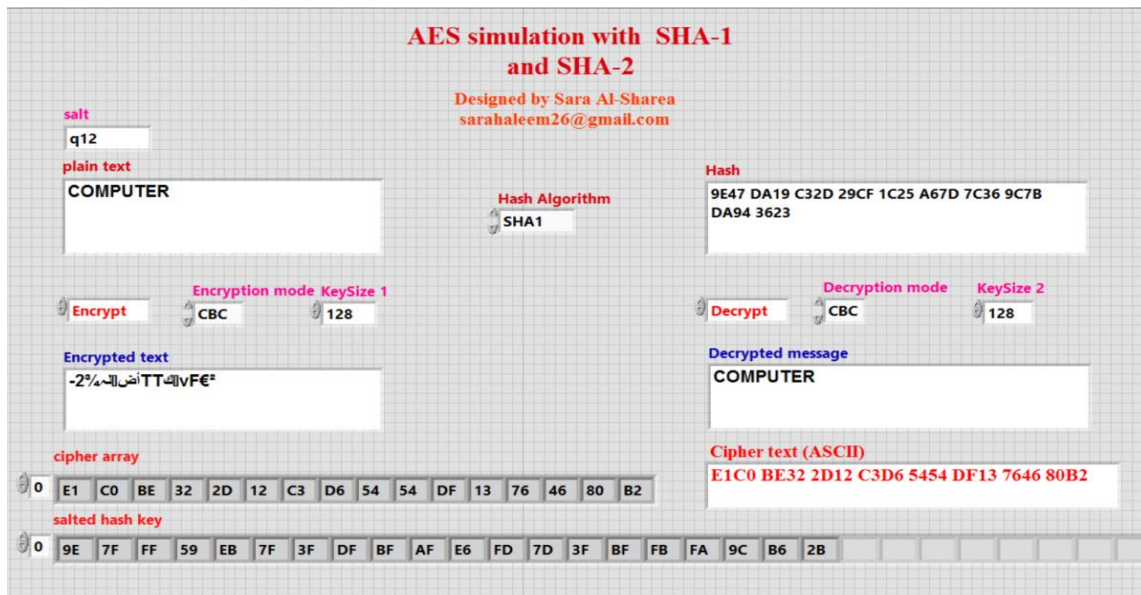


Figure 4.1: Testing AES and SHA-1 (capital letters)

The front panel of LabView in Figure 4.1 contains the input of the system (plain text) representing the string, which is typed as a text of different characters and numbers as illustrated previously to give the hash code represented by Hex values, which is converted from array to string and entered as a key to the AES. The cipher array (ciphertext) of the plain text "COMPUTER" is: E1C0 BE32 2D12 C3D6 5454 DF13 7646 80B2 which is 32 hex digit and represent the 192 bits of the AES cipher text. While the resulting hash code is: 9E47 DA19 C32D 29CF 1C25 A67D 7C36 9C7B DA94 3623 which is 32 hex digit and represent the 128 bits of the message digest. After decryption, the plaintext return the same as its value before encryption.

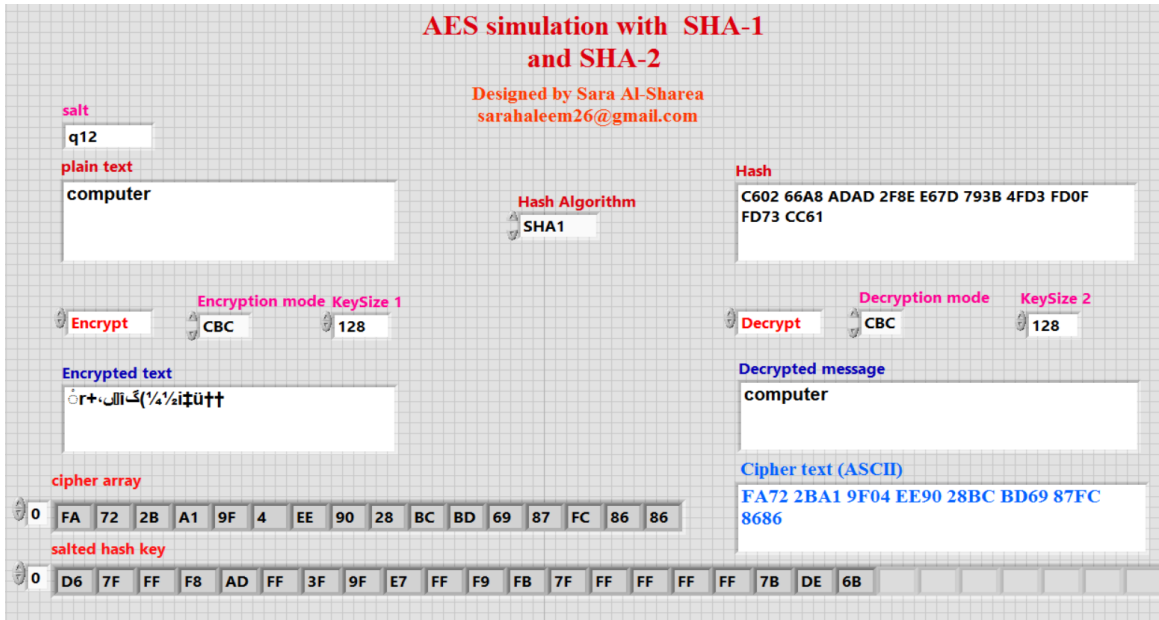


Figure 4.2: Testing AES and SHA-1 (small letters)

Figure 4.2 shows the changing the same text from the previous test, but with lower case. The hash code is completely different, showing the encrypted plain text is completely different and also the hash sensitivity to changes. The cipher array (ciphertext of the small same word "computer": FA72 2BA1 9F04 EE90 28BC BD69 87FC 8686 and the hash code is: C602 66A8 ADAD 2F8E E67D 793B 4FD3 FD0F FD73 CC61 and after decryption the plaintext return the same as its value before encryption.

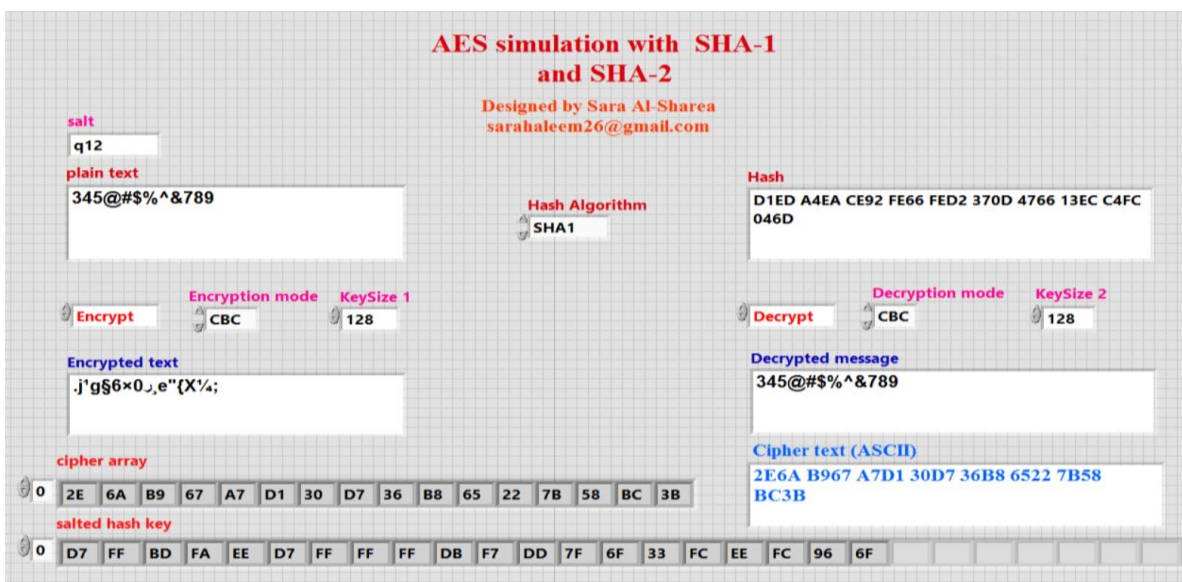


Figure 4.3: Testing AES and SHA-1 (symbols and numbers)

Figure 4.3 shows how the symbols and numbers are encrypted and how a simple change in the plain text entered will change all the hash code and the encrypted cipher text. The cipher array (ciphertext) of the entered text "345@#\$\$%^&789" is: 2E6A B967 A7D1 30D7 36B8 6522 7B58 BC3B and the hash code is: D1ED A4EA CE92 FE66 FED2 370D 4766 13EC C4FC 046D and after decryption the plaintext return the same as its value before encryption.

The same procedure is repeated with different modes such as electronic codebook mode, cipher feedback mode, output feedback mode and counter mode. The difference could be seen clearly in the ciphertext as shown below.

Figure 4.4 shows the difference in the ciphertext when entering the same word "COMPUTER" but choosing ECB mode instead of CBC mode in Figure 4.1.

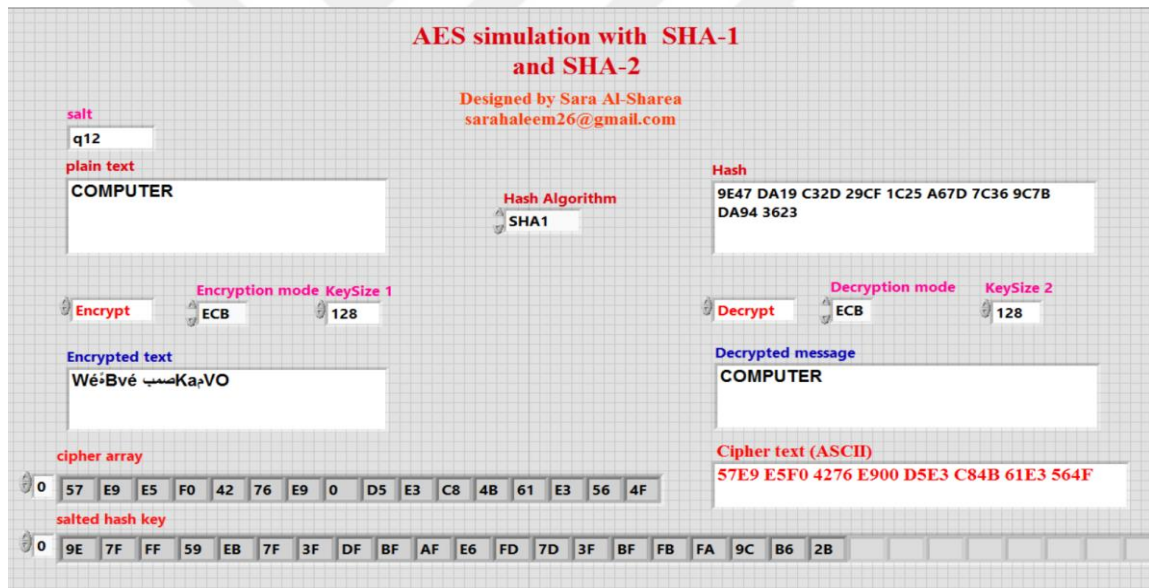


Figure 4.4: Testing AES and SHA-1 (ECB mode)

Figure 4.5 also shows the difference in the ciphertext, but when entering the same word "computer" in small letters and choosing CFB mode instead of CBC mode as in Figure 4.2. The same step was done for the other modes.

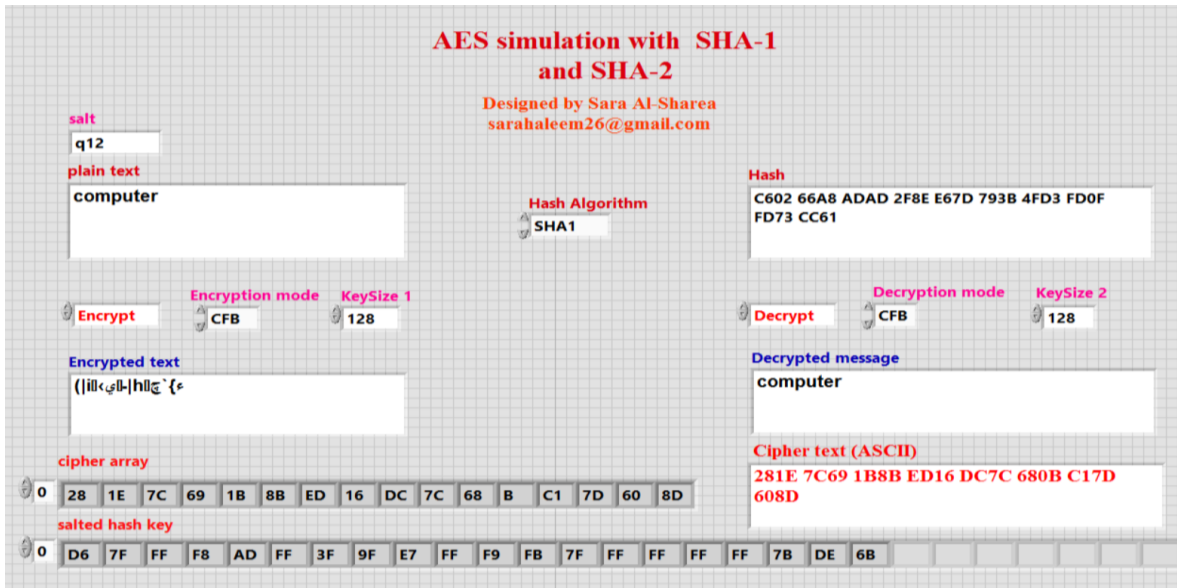


Figure 4.5: Testing AES and SHA-1 (CFB mode)

B. AES-256 and SHA-2 (256) Result by LabView

The front panel of LabView is shown in Figure 4.6, shows the text entered to the system "COMPUTER". The hash code of this text by SHA-2 is shown as Hash which is converted to string to be insert as the password to AES, the encrypted code of "COMPUTER" indicated as ciphertext, and after decryption the plaintext return the same as its value before encryption.

Figure 4.6 described the encryption of the following plain text "COMPUTER" is: 8982 0F00 D41A A34B 9CF9 72BD 7975 7BE2, which is 32 hex and represent the AES cipher text. The hash code is: B412 254D E584 8709 483A A99B B5D0 969A DF80 B676 8C87 B546 C460 41B0 0AB9 B858 which is 64 hex and represent the 256 bits message digest. This is done using CBC mode.

**AES simulation with SHA-1
and SHA-2**
Designed by Sara Al-Sharea
sarahaleem26@gmail.com

salt
q12

plain text
COMPUTER

Hash Algorithm
SHA256

Hash
B412 254D E584 8709 483A A99B B5D0 969A
DF80 B676 8C87 B546 C460 41B0 0AB9 B858

Encrypt Encryption mode KeySize 1
CBC 256

Decrypted message
COMPUTER

Decryption mode KeySize 2
CBC 256

Encrypted text
%٠٠٠للكEΚæür'/syu{â

Decrypted message
COMPUTER

Cipher text (ASCII)
8982 0F00 D41A A34B 9CF9 72BD 7975 7BE2

Cipher array
89 82 F 0 D4 1A A3 4B 9C F9 72 BD 79 75 7B E2

salted hash key
B5 D2 25 CD ED 9F BF 3D FA BF F9 9B BD FF BF FB DF EC F7 76 BF FF BD F6 F7 E1 51 B8 6E BD B8 7D

Figure 4.6: Testing AES-256 and SHA-2 (capital letters)

Figure 4.7 shows the encryption of the same plain text "computer" but in small letters. Thus it described how if we change a single letter will change the hash code and also the message encrypted. The encryption of the plain text "computer" is: 7761 98D8 9C9D 803E F352 5A3F FD73 A8E5 while the hash code is: AA97 3021 50FC E811 425C D845 3702 8A5A FBE3 7E3F 1362 AD45 A51D 467E 17AF DC9C and the decryption will return the plain text the same as before encryption.

**AES simulation with SHA-1
and SHA-2**
Designed by Sara Al-Sharea
sarahaleem26@gmail.com

salt
q12

plain text
computer

Hash Algorithm
SHA256

Hash
AA97 3021 50FC E811 425C D845 3702 8A5A FBE3
7E3F 1362 AD45 A51D 467E 17AF DC9C

Encrypt Encryption mode KeySize 1
CBC 256

Decrypted message
computer

Decryption mode KeySize 2
CBC 256

Encrypted text
wâbc€>RZ?s`

Decrypted message
computer

Cipher text (ASCII)
7761 98D8 9C9D 803E F352 5A3F FD73 A8E5

Cipher array
77 61 98 D8 9C 9D 80 3E F3 52 5A 3F FD 73 A8 E5

salted hash key
BB D7 31 ED D9 FF FA 35 FA FF D9 DD BF AF BF FB FF EF 7F 3F 33 7E BD F7 B7 DD 56 FE 7F AF DC BD

Figure 4.7: Testing AES-256 and SHA-2 (small letters)

Figure 4.8 shows how the symbols and numbers are encrypted and how a simple change in the plain text entered will change all the hash code and the encrypted cipher text. The encryption of the entered text "345@#\$\$%^&789" is: 0E00 8FD4 FBBD 0943 EDE3 E33C 254F 82B5 and the hash code is: 100B FB87 79A2 1799 C9EB F324 949F 26C9 3BB6 4A84 4C4F 3B75 53C9 6AD4 547D 2AC2 and after decryption the plaintext return the same as its value before encryption.

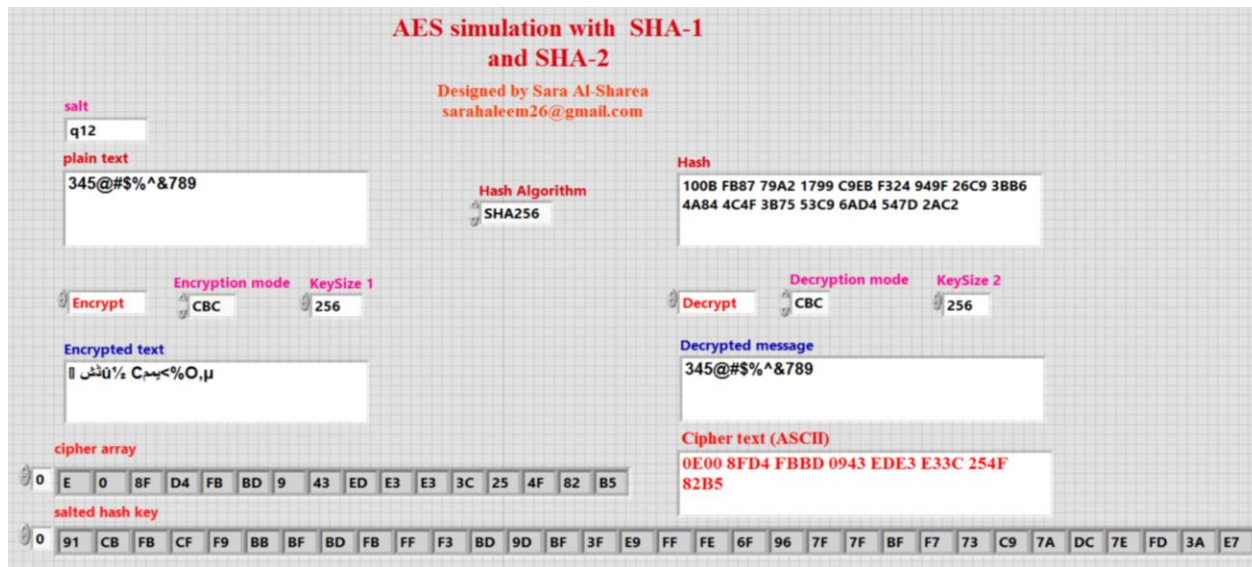


Figure 4.8: Testing AES-256 and SHA-2 (symbols and numbers)

The same procedure is repeated but with different modes as previously in SHA-1, the results are shown below.

Figure 4.9 shows the difference in the ciphertext when entering the same word "345@#\$\$%^&789" but choosing ECB mode instead of CBC mode in Figure 4.8.

AES simulation with SHA-1 and SHA-2

Designed by Sara Al-Sharea
sarahaleem26@gmail.com

salt
q12

plain text
345@#\$\$^&789

Hash Algorithm
SHA256

Hash
100B FB87 79A2 1799 C9EB F324 949F 26C9 3BB6
4A84 4C4F 3B75 53C9 6AD4 547D 2AC2

Encrypt Encryption mode: ECB KeySize 1: 256

Encrypted text
^:;vFRl L%#@:;!l

Decrypt Decryption mode: ECB KeySize 2: 256

Decrypted message
345@#\$\$^&789

cipher array
0 B2 A1 3B 76 66 52 5C 9 4C BE FD 40 8E 3A 8 A8

salted hash key
0 91 CB FB CF F9 8B BF BD FB FF F3 BD 9D BF 3F E9 FF FE 6F 96 7F 7F BF F7 73 C9 7A DC 7E FD 3A E7

Figure 4.9: Testing AES-256 and SHA-2 (ECB mode)

Figure 4.10 also describes the difference in the ciphertext, but when entering the same word "computer" in small letters and choosing CFB mode instead of CBC mode as in Figure 4.7.

AES simulation with SHA-1 and SHA-2

Designed by Sara Al-Sharea
sarahaleem26@gmail.com

salt
q12

plain text
computer

Hash Algorithm
SHA256

Hash
AA97 3021 50FC E811 425C D845 3702 8A5A FBE3
7E3F 1362 AD45 A51D 467E 17AF DC9C

Encrypt Encryption mode: CFB KeySize 1: 256

Encrypted text
1'k%ll+ljZ^'n^z

Decrypt Decryption mode: CFB KeySize 2: 256

Decrypted message
computer

cipher array
0 31 88 6B BC 3 F7 49 D2 5A 5E A8 6E C7 8A 5E BA

salted hash key
0 BB D7 31 ED D9 FF FA 35 FA FF D9 DD BF AF BF FB FF EF 7F 3F 33 7E BD F7 B7 DD 56 FE 7F AF DC BD

Figure 4.10: Testing AES-256 and SHA-2 (CFB mode)

4.3 THE IMPLEMENTATION OF THE PROPOSED ALGORITHM BASED ON FPGA

To import the written VHDL code of the proposed algorithms to System Generator, the Xilinx Black Box is used as illustrated in Figure 4.11. After that, the proposed algorithm was implemented and simulated by MATLAB/Simulink connected with ISE 14.7. The **clock** input is the input system frequency to the black box that shows the validity of the input data, the **Data in** is the plain text entered to be encrypted which consists of 128 bits, the **Key** is the chosen input key (SHA output code), while the **Cipher** is the output of the proposed algorithm.

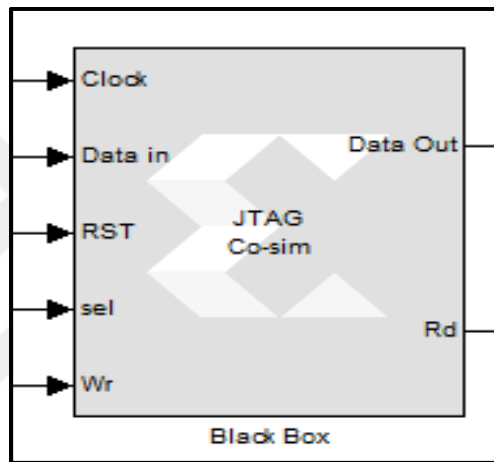


Figure 4.11: Proposed System Black Box

In the output of the proposed system, the reverse transformation is important to put into words the system generator data type to every wanted data kind held by MATLAB. Xilinx 'Gateway Out' is blocks assigned to the output Cipher which represents the ciphered data and used to complete the operation.

Figure 4.12 shows the encryption process of the proposed system, the same steps and the same black box is shown in the decryption process in Figure 4.13. The input of the black box is the ciphered data and the output is the original data.

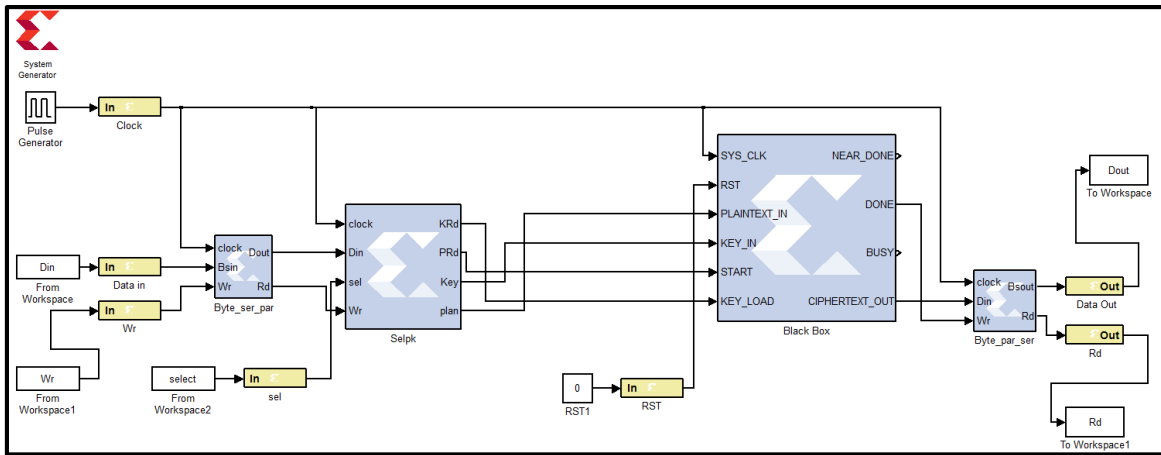


Figure 4.12: Encryption Process of The Proposed System

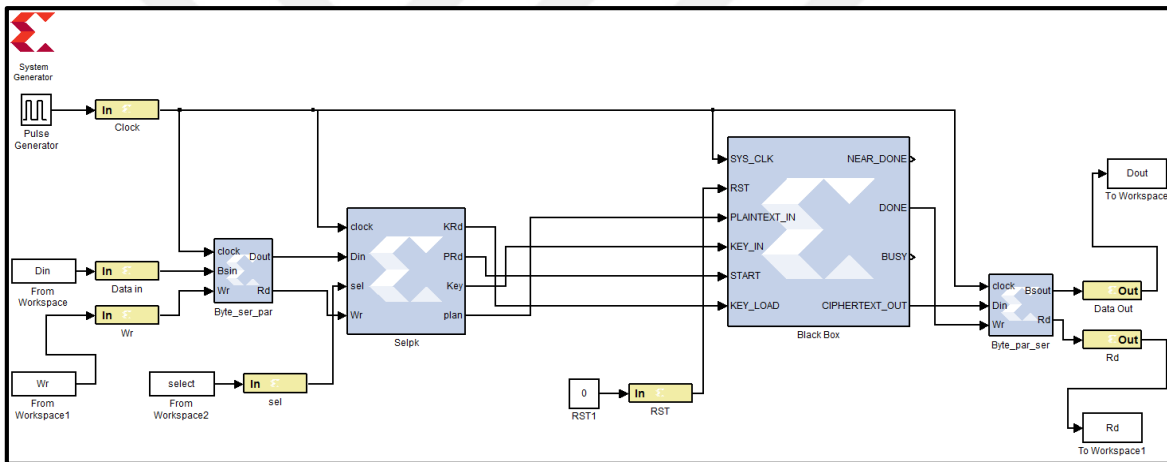
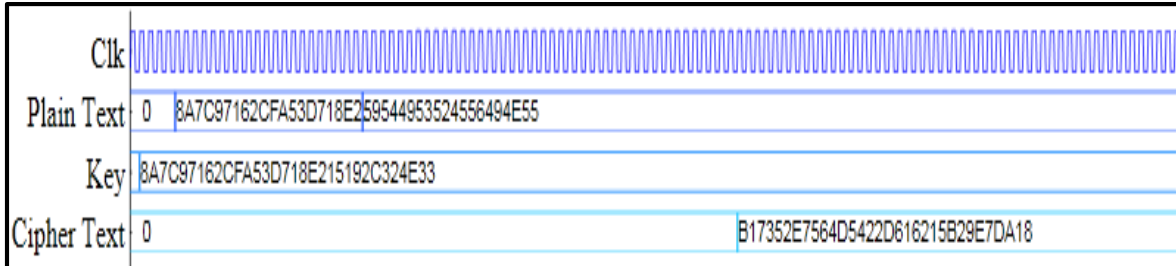


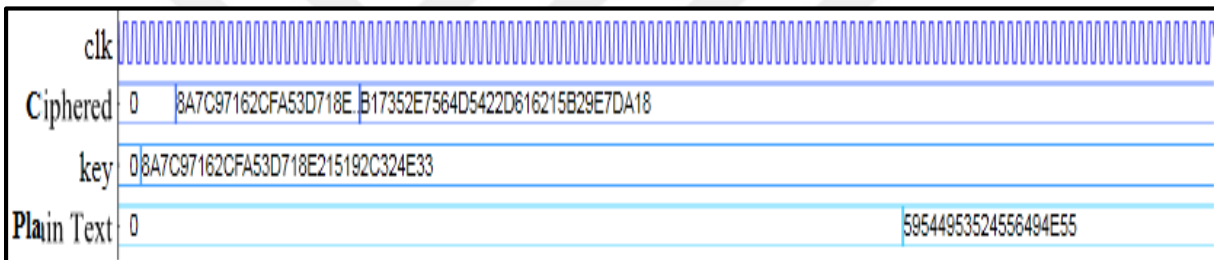
Figure 4.13: Decryption process of The Proposed System

Figure 4.14 illustrates the simulation result of encryption and decryption of SHA-1 and AES by using Xilinx 14.7 simulator. It shows that input plaintext that is represented in hex Number entered to be encrypted, and the input key that is represented in hex Number which is generated by SHA-1 to be the key to the AES algorithm. The key load signal must be loaded before loading the signal of plaintext start, so as to prepare the key required in the top round then the load signal will be activated to load the plaintext. Signal (BUSY) means that if the system is really work or not, it will be at the first time equal to (0) until the starting of the first round, then it will be kept activated until the system complete the encryption process. Signal (NEAR_DONE) is active when the system is near to finish the encryption process. Signal (DONE) is active when system

completes the encryption process and displays the final result 'ciphertext'. The output of the simulation decryption process is the same plaintext of the case study which shows that the system runs successfully.



(a) Simulation of the encryption process that done by using SHA-1 with AES



(b) Simulation of the decryption process that done by using SHA-1 with AES

Figure 4.14: Simulation Output of the Encryption and Decryption Process

4.4 MAPPING THE PROPOSED ALGORITHM ON FPGA

In order to achieve the design flow of the FPGA, four stages are done. These stages are: the design entry, the design synthesis, the design implementation and finally the Xilinx device programming. The structure of the suggested system contains writing the VHDL code of the system in the design entry stage. Design hierarchy is checked and analyzed by the VHDL code arrangement in order to make the best or most effective use of the proposed system design to FPGA architecture that was completed in the design synthesis stage. The design execution step is in charge of:

- a. By "Translate" process, merge input net lists and restriction inside design files

b. By "Map" process, fit framework design to the references of the FPGA hardware device

c. By "Place and Route" process, place and route framework design with the timing restrictions.

In the step of the "Xilinx device programming", the bits stream file would be generated for the system design and to be loaded to the Xilinx device.

Table 4.1 shows the design summary of algorithm in both encryption and decryption process which takes 22% of the hardware device slices of Spartan3A 700AN slices as it is clear in the device utilization summary below.

Table 4. 1: Encryption and Decryption Process Utilization

(a) Encryption Process Utilization Summary

| Device Utilization Summary | | | |
|---|-------|-----------|-------------|
| Logic Utilization | Used | Available | Utilization |
| Number of slice flip floops | 1,710 | 11,776 | 14% |
| Number of 4input LUTS | 1,135 | 11,776 | 9% |
| Number of occupid slices | 1,337 | 5,888 | 22% |
| Number of slices containig only related logic | 1,337 | 1,337 | 100% |
| Number of slices containig un related logic | 0 | 1,337 | 0% |
| Total Number of 4 input LUTS | 1,135 | 11,776 | 9% |
| Number of bounded IOBs | 21 | 372 | 5% |
| IOB Flip Flop | 9 | | |
| Number of BUFGMUXs | 1 | 24 | 4% |
| Number of RAMB 16BWFs | 10 | 20 | 50% |
| Average Fanout of Non-Clock Nets | 3.76 | | |

(b) Decryption Process Utilization Summary

| Device Utilization Summary | | | |
|---|-------|-----------|-------------|
| Logic Utilization | Used | Available | Utilization |
| Number of slice flip floops | 1,710 | 11,776 | 29% |
| Number of 4input LUTS | 1,135 | 11,776 | 22% |
| Number of occupid slices | 1,337 | 5,888 | 53% |
| Number of slices containig only related logic | 1,337 | 1,337 | 100% |
| Number of slices containig un related logic | 0 | 1,337 | 0% |
| Total Number of 4 input LUTS | 2,814 | 11,776 | 23% |
| Number used as logic | 2,686 | | |
| Number used as a route-thru | 128 | | |
| Number of bounded IOBs | 21 | 372 | 5% |
| IOB Flip Flop | 9 | | |
| Number of BUFGMUXs | 1 | 24 | 4% |
| Number of RAMB 16BWFs | 10 | 20 | 50% |
| Average Fanout of Non-Clock Nets | 3,43 | | |

After the system design pass into the FPGA design flow steps successfully, a file of the proposed algorithms is created called bits stream, and it is ready to be downloaded into the hardware platform. The bits stream is downloaded into the hardware platform and it will configure the hardware device according to the wanted system design. Different programs are utilized to download the "bits stream" file to an FPGA device, such as the XILINX ISE 14.7 that configures the target device process in order to transfer the bits stream via JTAG cable to the FPGA program. Another programs such as hardware co-simulator system generator in MATLAB can be used, to configure the device and link the proposed design operation in the FPGA directly into a MATLAB/Simulink simulation. So, utilizing the HW Co-simulation is useful in this work

since it is significant to check the hardware program by transferring the input data to the device in addition to the bits stream file and lock onto the achievement.

4.5 THE PROPOSED SYSTEM IMPLEMENTATION ON SPARTAN-3A 700AN PLATFORM

After the steps of the FPGA design flow has done successfully, a bits stream file of the proposed system would be made and prepared to download into the hardware program. The bits stream is downloaded in order to configure the hardware device according to the wanted system design. Several software programs were used to download the file into the device of the FPGA; XILINX ISE 14.7 is one of the softwares that was used. It configures the wanted device process in order to transfer the bits stream by the JTAG cable to the FPGA program. Another package such as the MATLAB, it used the system generator hardware co-simulator in order to define the device and combine the proposed system design working in the FPGA at once into a MATLAB/Simulink simulation. As a consequence, utilizing the HW Co-simulation was very useful in this system since it is significant to check the hardware program by transferring the input data and the bits stream file to the device and get the outcomes.

The connection that supports the hardware device is either Ethernet or JTAG cable relying on the accessibility of these cables. A JTAG cable is used in this work to define the proposed algorithm on the FPGA board, shown in Figure 4.15.



Figure 4.15: JTAG Cable

The comparison between the results of the FPGA system design simulation and the hardware platform shows both results are the same during the execution for all tests when the run button of the MATLAB is hilted. The HW Co-simulation (hwcosim) was putted with the FPGA design of the proposed system. When the Simulink simulation begun, the bits stream would be

downloaded to the hardware device in order to define it as seen in Figure 4.16. When the device configuration was finished, the link between hwcosim and the FPGA device would be re-established so as to begin the co-simulating of the system design as shown in Figure 4.17.

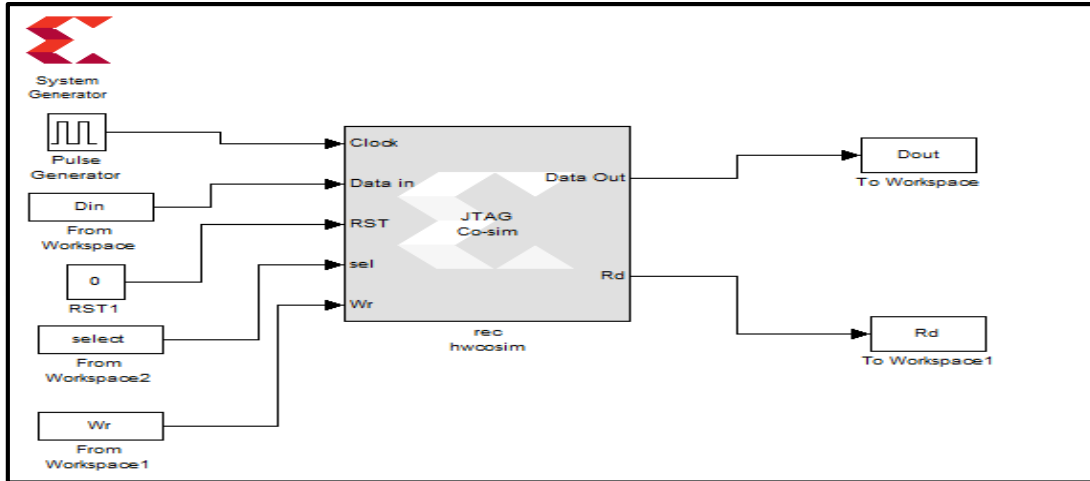


Figure 4.16: Hardware Co-simulation

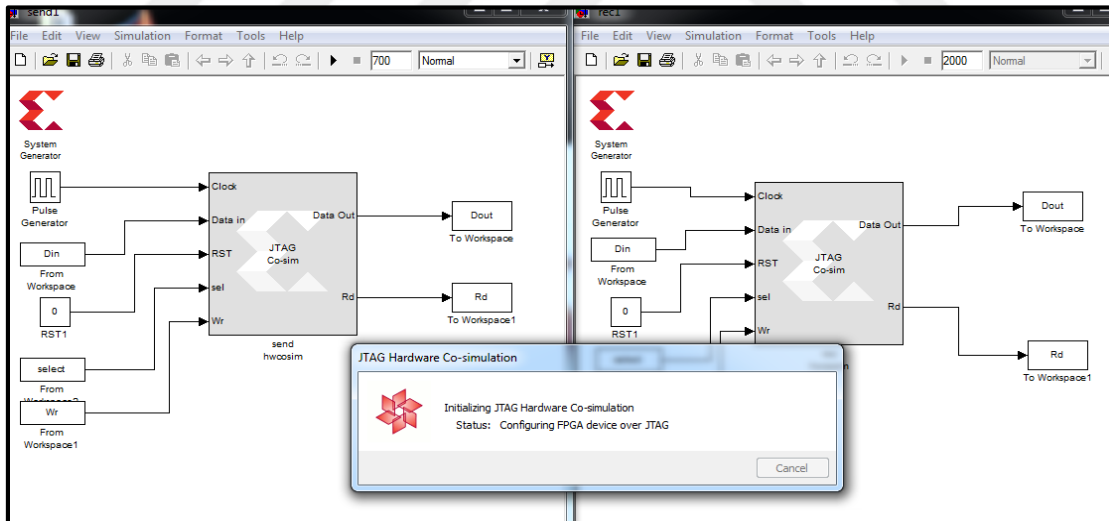


Figure 4.17: The bits stream downloading to the hardware device through the JTAG cable

Figure 4.18 shows how the connection was done between the computer and Spartan-3A 700AN board that was held by the proposed system design.

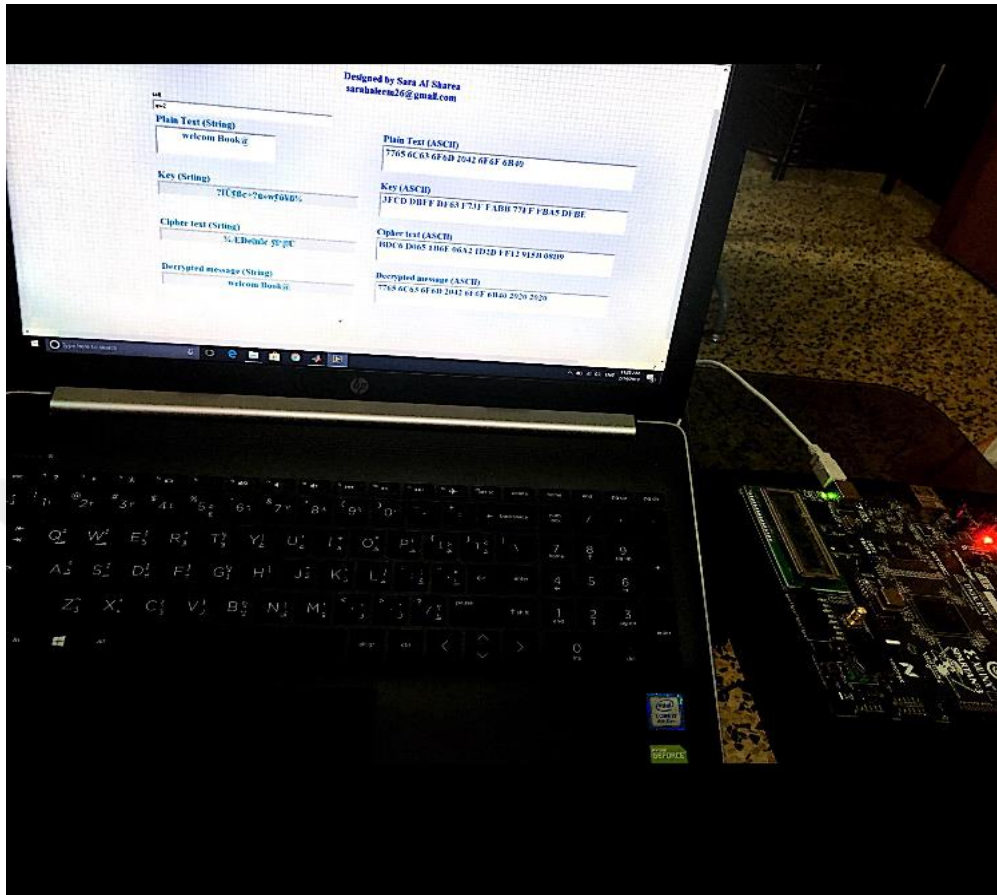


Figure 4.18: Connection between the computer and Spartan-3A 700AN board

4.6 STATISTICAL TESTS

A lot of testing techniques were utilized to determine encryption quality and system execution. The result of the testing technique is done by testing the randomness attitude using the FIPS PUB 140-1 statistical tests.

Table 4.2 displays the result of accomplishing the FIPS PUB 140-1 statistical tests in order to examine the system randomness. The results shown in the Table conclude that the proposed algorithm system had achieved the randomness requirements needed for safe encryption algorithm.

Table 4.2: FIPS PUB 140-1 statistical tests of the proposed system

| Statistical test | Freedom Degree | Test result of proposed algorithm | Pass/Fail | |
|-----------------------------|-----------------------|-----------------------------------|-----------|------|
| Frequency test | MUST BE \leq 11.81 | 1.125 | PASS | |
| Serial test | MUST BE \leq 7.81 | 5.625 | PASS | |
| Poker test | MUST BE \leq 11.1 | 5.850 | PASS | |
| Run test | MUST BE \leq 10.788 | 2.500 | PASS | |
| Autocorrelation test | SHIFT NO. 1 | MUST BE \leq 3.84 | 1.331 | PASS |
| | SHIFT NO. 2 | | 3.841 | FAIL |
| | SHIFT NO. 3 | | 5.000 | FAIL |
| | SHIFT NO. 4 | | 1.581 | PASS |
| | SHIFT NO. 5 | | 0.073 | PASS |
| | SHIFT NO. 6 | | 0.033 | PASS |
| | SHIFT NO. 7 | | 23.215 | FAIL |
| | SHIFT NO. 8 | | 0.833 | PASS |
| | SHIFT NO. 9 | | 8.076 | FAIL |
| | SHIFTNO. 10 | | 4.881 | FAIL |

Table 4.2 shows the results of the FIPS PUB 140-1 statistical tests. Every test has a special condition and a freedom degree, where the frequency test accomplished a success rate that exceeded the condition, and crossed the wall. The goal of the frequency test is to calculate whether the number of 0's and 1's in s is almost the same as would be expected for the random sequence in reference a certain freedom degree. In the serial test, the suggested algorithm system also succeeded. The goal of this test is to determine whether the number of occurrences of 00, 01, 10, and 11 as subsequences of s are almost the same as would be expected for the random sequence.

The proposed encryption algorithm could pass the Pocker and Run test since it achieved the condition. As in the Poker test, the proposed system achieved whether the sequences of stream length appear almost the same number of time as prospective for a random sequence. And in the Run test the proposed system achieved a good success since the condition of this test calculate whether the number of runs of either zeros or ones of different lengths in the sequences is as expected for the random sequence that it is easy to pass through the proposed system.

Many results are there for the Autocorrelation test, it achieved success in most results, while failing in some of them, since it relies on checking of the correlations between the sequence of bits stream and the shifted versions of it, and this is probably not always accomplished in the sequence bits stream of the proposed system.



5. CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORKS

5.1 CONCLUSIONS

From the previous chapters and the results from the proposed technique, the most important concluded points that achieved could be summarized as follows:

1. From the proposed idea, the confidentiality achieved by the combination of one way encryption technique (Hash Algorithms) as a key and two way encryption technique (AES Algorithm) to get integrity and authority with high power security .
2. All implementations of the proposed security system based on simple GUI software as the human machine interaction, which is simply modified the algorithm by changing the block parameters, also obtain the signal flow through the stages of the proposed system software implementation.
3. From the simulation results of the proposed system implementation good results obtained and different text level tested into the proposed security system (capital, small letters, symbols, numbers) with different sizes from (1-bits to 2256 bits).
4. Different security and statistical testing techniques were applied to the proposed system, in order to obtain the power of security, which was a 90% success percentage.
5. Depending on the testing results in software the hardware implementation of the proposed system (AES with SHA-1) had been accomplished based on FPGA card. The approximation same results obtained with respect to simulation results offers less processing times as compared with software processing time. The results of the hardware design are harmonic with simulation results and this leads to the strength of the work and trustiness.
6. In hardware, the proposed system takes less than 22% of hardware device slices in encryption and less than 53% of hardware device slices in decryption.

5.2 FUTURE WORK

A number of future projects can be suggested to develop the proposed work as stated below:

1. Implementation of more powerful security algorithm of modified AES with complicated hash algorithms in both software and hardware using another GUI software and another hardware kit such as raspberry or other hardware kits
2. Using signal flow graph technique to reduce the size of the hardware in FPGA card and in iteration of both AES and Hash algorithms which deals with some simple blocks in iterative form
3. Build the most common (commands, word, numbers) in some application and create a security reference dictionary of the hash key to avoid the collision in keys because the limited size of the hash algorithms.
4. Build the complete database depending on the proposed system for generation high level security system.

REFERENCES

- [1] Hans Delfs and Helmut Knebl, "Introduction to Cryptography Principles and Applications", 3rd ed. Berlin Heidelberg, New York Dordrecht London: Springer-Verlag Berlin Heidelberg, 2006.
- [2] Razi Hosseinkhani and Seyyed Hamid Haj Seyyed Javadi, "Using image as cipher key in AES ," IJCSI International Journal of Computer Science Issues, vol. 9, no. 2, pp. 538-544, March 2012.
- [3] Ashwini M. Deshpande, Mangesh S. Deshpande, and Devendra N. Kayatanavar, "FPGA Implementation of AES Encryption and Decryption," International Conference on "Control, Automation, Communication And Energy Conservation, June 2009.
- [4] Pravin B. Ghwari, Mrs. Jaymala K. Patil, and Amit B. Chougule, "Efficient Hardware Design and Implementation of AES Cryptosystem", International Journal of Engineering Science and Technology, vol. 2(3), pp. 213-219, 2010.
- [5] William Stallings, "Cryptography and Network Security Principles and Practice, Fifth Edition" ed., Rose Kernan et al., Eds., 2011, 2006.
- [6] Rubina B. Patel and Naveen Chaudhary , "Special Issue of International Journal of Computer Applications (0975 – 8887) on Communication Security," Analyzing Digital Signature Robustness with Message Digest Algorithms Digest Algorithms, pp. 41-44, Mar 2012.
- [7] M. Pitchaiah, Philemon Danie, and Praveen , "Implementation of Advanced Encryption Standard Algorithm," International Journal of Scientific & Engineering Research, vol. 3, no. 3, pp. 1-6, March 2012.
- [8] Richa Sharma, Purnima Gehlot, and S. R. Biradar, "VHDL Implementation of AES Algorithm".
- [9] Raaed K. Ibrahim, Ali SH. Hussain, and Roula A. Kadhim, "Implementation of Secure Hash Algorithm SHA-1 by Labview," International Journal of Computer Science and Mobile Computing, vol. 4, no. 3, pp. 61-67, March 2015.
- [10] Sagar Paddhan, Padma Lohiya, and Sudhir Shelke, "International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering," AES-256 Encryption in Communication using LabVIEW, vol. 4, no. 6, pp. 533-5340, June 2015.

- [11] Lawrence C. Washington, "Introduction to Cryptography". Wade Trappe: Wireless Information Network Laboratory, 2006.
- [12] (2016) Stanford Encyclopedia of Philosophy. [Online]. <https://plato.stanford.edu/entries/computational-complexity/>
- [13] Bruce Schneier, "Applied Cryptography", 2nd ed. Mountain View, CA: Whitfield Diffie.
- [14] (2017) E-Power. [Online]. <http://developer.e-power.com.kh/one-way-vs-two-way-encryption/>
- [15] William Stallings, "Cryptography and Network Security Principles and Practices, Fourth Edition". United States of America: Prentice Hall, 2005.
- [16] Ioannis Yiakoumis, Markos Papadonikolakis, Harris Michail, Athanasios P. Kakarountas, and Costas E. Goutis, "Efficient Small-Sized Implementation of the Keyed-Hash Message Authentication Code," EUROCON, the International Conference, pp. 1875-1878, 2005.
- [17] A. Menezes, P. van Oorschot, and S. Vanstone, Handbook of Applied Cryptography.: Massachusetts Institute of Technology (MIT), 1996.
- [18] Marc Martinus Jacobus Stevens, "Attacks on Hash Functions and Applications". Amsterdam: The thesis carried out at the Centrum Wiskunde & Informatica in the Netherlands and has been funded by the NWO VICI grant of Prof. dr. R. Cramer. Printed by Ipskamp Drukkers, AMS 2000 Subj. class. code: 94A60, ISBN: 978-94-6191-317-3, Universiteit Leiden, 2012.
- [19] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. , Handbook of Applied Cryptography., 1996.
- [20] Rubina B. and Patel Naveen Chaudhary, "Analyzing digital signature robustness with message digest algorithm," IJCA Special Issue on Communication Security, 2012.
- [21] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone, "Handbook of Applied Cryptography". United State of America: Massachusetts Institute of Technology MIT, 1996.
- [22] (2014, November) WhatIs.com. [Online]. <http://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA>
- [23] Douglas Selent, "Advanced Encryption Standard," River Academic Journal, vol. 6, pp. 1-14, 2010.

- [24] M.Pitchaiah, Philemon Daniel, and Praveen, "Implementation of Advanced Encryption Standard algorithm," International Journal of Scientific & Engineering Research, vol. 3, no. 3, pp. 1-6, March 2012.
- [25] J. Daemen and V. Rijmen, "The block cipher Rijndael," Smart Card research and Applications ,LNCS 1820, Springer Verlag, pp. 88-296.
- [26] Bibek Bhattarai, Bipin Thapa Magar, Naresh Kumar Giri, and Gaurav Sitaula, "FPGA Prototyping of the secured biometric based Identification system," ResearchGate, 2014.
- [27] M. Pitchaiah, Philemon Daniel, and Praveen , "Implementation of Advanced Encryption Standard Algorithm," International Journal of Scientific & Engineering Research, vol. 3, no. 3, pp. 2229-5518, march 2012.
- [28] Bart Preneel, "Analysis and Design of Cryptographic Hash Functions," pp. 1-323, February 2003.
- [29] "<http://www.metamorphosite.com/one-way-hash-encryption-sha1-data-software>," pp. 1-13, accessed on 30/12/2016.
- [30] Atul Kahate, "Cryptography and Network Security.Second Edition," 2003, Tata McGraw-Hill Pvt.
- [31] Ivan Damgård, "A design principle for hash functions," In Gilles Brassard, editor, Advances in Cryptology: CRYPTO 89, volume 435, pp. 416-427, 1989.
- [32] Ralph Merkle, "One way hash functions and DES," In Gilles Brassard,editor, Advances in cryptology: CRYPTO 89, volume 435, pp. 428-446, 1989.
- [33] Murali Krishna Reddy Danda, "Design and Analysis of Hash Function"., 2007.
- [34] D. Zibin and Z. Ning, "FPGA Implementation of SHA-1 Algorithm," 0-7803-7889-X/03/\$17.00@2003 IEEE, pp. 1321-1324, 2003.
- [35] "<https://www.tbs-certificates.co.uk/FAQ/en/sha256.html>," last accessed on 2-1-2017.
- [36] C. i Xiao-hu and D. Jian-zhi, "Design of SHA-1 Algorithm based on FPGA," International Conference on Networks Security, Wireless Communications and Trusted Computing IEEE, pp. 532-534, 2010.
- [37] Ryan Glabba, Laurent Imbertb, and Graham Jullien, "Multi-mode operator for SHA-2 hash functions," Journal of Systems Architecture, vol. 53, no. 2-3, pp. 12-138, March 2007.

- [38] R. V. Mankar and Prof. S. I. Nipanikar, "C Implementation of SHA-256 Algorithm," International Journal of Emerging Technology and Advanced Engineering, vol. 3, no. 6, pp. 167-170, June 2013.
- [39] Mohit Arora. (2012, July) EETimes. [Online]. http://www.eetimes.com/document.asp?doc_id=1279619
- [40] Mohammad A. AlAhmad and Imad Fakhri Alshaikhli, "Broad View of Cryptographic Hash Functions," IJCSI International Journal of Computer Science Issues, Vol. 10, Issue 4, No 1, pp. 239-246, 2013.
- [41] Blondeau, Thomas Peyrin, and Lei Wang, "Known-key Distinguisher on Full PRESENT," Known-key Distinguisher on Full PRESENT, pp. 1-20.
- [42] Eli Biham, "New types of cryptanalytic attacks using related keys," Springer-Verlag Berlin Heidelberg 1994, pp. 398-409, 1994.
- [43] P. Burns, "Software Defined Radio for 3G", 1st ed. LONDON, 2003.
- [44] T. L. Floyd, "Digital Fundamentals", 9th ed. USA, 2006.
- [45] Quora. [Online]. <https://www.quora.com/What-is-the-difference-between-CPLD-and-FPGA>
- [46] JuanSoto,JamesNechvatal,Milesmid,ElaineBarker,StefanLeigh, MarkLevenson, Mark Vangel, DavidBanks, AlanHeckert, JamesDray, SanVo AndrewRukhin, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," USA, 2010.
- [47] Dr Alaa kadhim and Mohammed salih, "Proposal of New Keys Generator for DES Algorithms Depending on Multi Techniques," Eng. &Tech., vol. 32, pp. 1-13, 2014.

APPENDIX A

S-BOX OF THE AES

The S-box depended on the AES algorithm that used in the section (2.3.1) is shown:-

Table A.1: The Values of the S-Box

| | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 118 | 171 | 215 | 254 | 43 | 103 | 1 | 48 | 197 | 111 | 107 | 242 | 123 | 119 | 124 | 99 |
| 192 | 114 | 164 | 156 | 175 | 162 | 212 | 173 | 240 | 71 | 89 | 25 | 125 | 201 | 130 | 202 |
| 21 | 49 | 216 | 113 | 241 | 229 | 165 | 52 | 204 | 247 | 63 | 54 | 38 | 147 | 253 | 183 |
| 117 | 178 | 39 | 235 | 226 | 128 | 18 | 7 | 154 | 5 | 150 | 24 | 195 | 35 | 199 | 4 |
| 132 | 47 | 227 | 41 | 179 | 214 | 59 | 82 | 160 | 90 | 110 | 27 | 26 | 44 | 131 | 9 |
| 207 | 88 | 76 | 74 | 57 | 190 | 203 | 106 | 91 | 177 | 252 | 32 | 237 | 0 | 209 | 83 |
| 168 | 159 | 60 | 80 | 127 | 2 | 249 | 69 | 133 | 51 | 77 | 67 | 251 | 170 | 239 | 208 |
| 210 | 243 | 255 | 16 | 33 | 218 | 182 | 188 | 245 | 56 | 157 | 146 | 143 | 64 | 163 | 81 |
| 115 | 25 | 93 | 100 | 61 | 126 | 167 | 196 | 23 | 68 | 151 | 95 | 236 | 19 | 12 | 205 |
| 219 | 11 | 94 | 222 | 20 | 184 | 238 | 70 | 136 | 144 | 42 | 34 | 220 | 79 | 129 | 96 |
| 121 | 228 | 149 | 145 | 98 | 172 | 211 | 194 | 92 | 36 | 6 | 73 | 10 | 58 | 50 | 224 |
| 8 | 174 | 122 | 101 | 234 | 244 | 86 | 108 | 169 | 78 | 213 | 141 | 109 | 55 | 200 | 231 |
| 138 | 139 | 189 | 75 | 31 | 116 | 221 | 232 | 198 | 180 | 166 | 28 | 46 | 37 | 120 | 186 |
| 158 | 29 | 193 | 134 | 185 | 87 | 53 | 97 | 14 | 246 | 3 | 72 | 102 | 181 | 62 | 112 |
| 223 | 40 | 85 | 206 | 233 | 135 | 30 | 155 | 148 | 142 | 217 | 17 | 17 | 152 | 248 | 225 |
| 22 | 187 | 84 | 176 | 15 | 45 | 153 | 65 | 104 | 66 | 230 | 191 | 13 | 137 | 161 | 140 |

Table A.2: The Inverted Values of the S-Box

| | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 251 | 215 | 243 | 129 | 158 | 163 | 64 | 191 | 56 | 165 | 54 | 48 | 213 | 106 | 9 | 82 |
| 203 | 233 | 222 | 196 | 68 | 67 | 142 | 52 | 135 | 255 | 47 | 155 | 130 | 57 | 227 | 124 |
| 78 | 195 | 250 | 66 | 11 | 149 | 76 | 238 | 61 | 35 | 194 | 166 | 50 | 148 | 123 | 84 |
| 37 | 209 | 139 | 109 | 73 | 162 | 91 | 118 | 178 | 36 | 217 | 40 | 102 | 161 | 46 | 8 |
| 146 | 182 | 101 | 93 | 204 | 92 | 164 | 212 | 22 | 152 | 104 | 134 | 100 | 246 | 248 | 114 |
| 132 | 157 | 141 | 167 | 87 | 70 | 21 | 94 | 218 | 185 | 237 | 253 | 80 | 72 | 112 | 108 |
| 6 | 69 | 179 | 184 | 5 | 88 | 228 | 247 | 10 | 211 | 188 | 140 | 0 | 171 | 216 | 144 |
| 107 | 138 | 19 | 1 | 3 | 189 | 175 | 193 | 2 | 15 | 63 | 202 | 143 | 30 | 44 | 208 |
| 115 | 230 | 180 | 240 | 206 | 207 | 242 | 151 | 234 | 220 | 103 | 79 | 65 | 17 | 145 | 58 |
| 110 | 223 | 117 | 28 | 232 | 55 | 249 | 226 | 133 | 53 | 173 | 231 | 34 | 116 | 172 | 150 |
| 27 | 190 | 24 | 170 | 14 | 98 | 183 | 111 | 137 | 197 | 41 | 29 | 113 | 26 | 241 | 71 |
| 244 | 90 | 205 | 120 | 254 | 192 | 219 | 154 | 32 | 121 | 210 | 198 | 75 | 62 | 86 | 252 |
| 95 | 236 | 128 | 39 | 89 | 16 | 18 | 177 | 49 | 199 | 7 | 136 | 51 | 168 | 221 | 31 |
| 239 | 156 | 201 | 147 | 159 | 122 | 229 | 45 | 13 | 74 | 181 | 25 | 169 | 127 | 81 | 96 |
| 97 | 153 | 83 | 131 | 60 | 187 | 235 | 2 | 176 | 245 | 42 | 174 | 77 | 59 | 224 | 160 |
| 125 | 12 | 33 | 85 | 99 | 20 | 105 | 225 | 38 | 214 | 119 | 186 | 126 | 4 | 43 | 23 |

APPENDIX B

HASH CALCULATOR

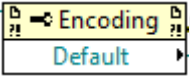
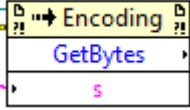
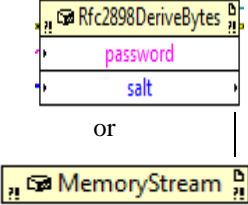


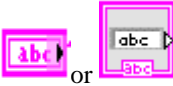
| Message | Hash value/SHA-1 | Hash value'SHA-2 |
|-----------|--|--|
| Array | 1070 0447 2B7BA4E5 E31F 3082 EE1F B5A1 239E EC61 | F0CF 39D0 BE3E FBB6 F86A C240 4100 FF7E 055C 17DE D946 A068 08D6 6F89 CA03 A811 |
| Generator | 1D20 DE03 126B 297E 05C1 3A7D 280F 33E2 4C72 C537 | 1ADE 60AB E93F 5CFC E908 A10B B2D1 B474 F024 7798 43FA BA4B B1F2 EF06 F277 D3C7 |
| Evening | 6734 E197 90F1 1127 0C9E 4D31 EAC3 1CEF 4653 4E85 | 458C 1FED 5743 5404 2397 E664 D6DA BE58 415C 8AB6 29F0 52DB 5723 FB42 B490 D280 |
| money | C952 59DE 1FD7 1981 4DAE F8F1 DC4B D64F 9D88 5FF0 | 8D2A C8B5 8EAD 9744 D772 86DE 9B0B CB7A 894F 238C 3149 FC9F 3B1E 3CAF F363 30FE |
| Submitted | FDA4 3783 BE8E 0B3F 9206 A57F 2248 C223 7F04 9F81 | E06F 8CFC 39F1 2AA9 8DFE BC04 587E D1A5 CF4D 6588 9EC9 B02D 4CF4 B9B9 4DF2 2E69 |
| System | BC07 92D8 DC81 E8AA 30B9 8724 6A5C E97C 40CD 6833 | 6725 E7BB CD28 F3A8 A586 FA34 BF19 1FD7 2DDE 8B61 7569 32CD 3237 C17A 6F19 6F1A |
| Perform | 49B7 8AF2 D644 590F F6F7 3D4D 7C64 E821 EE5B 4E08 | 6ADB 770A 21AE 7200 BFAC 6D1E 694F C76B 3E8B 8166 C788 0223 62ED 1E66 055A B88A |
| Discover | 4827 EA22 716A 74AA F8BB E499 FC31 4BB9 3F73 C65E | D4A3 3D5B 78BC CEBE 3F16 843D C30E 6C0F 73B4 EB6E |






| | | |
|----------------------|---|--|
| | | FB4E 7114 DDFE BDE7 FA2C 9954 |
| What do you need? | 2EF8 75B9 BF67 843A C034 CA44 671C 7357 F602 55B4 | 52C6 A4F6 8307 D05B FD64 60D1 DD80 7E2F A05D 65B9 4FEF EDB7 4C63 296B 3C78 642A |
| How much it's cost? | 7C98 1020 D9D75F09 0317 A244 67DE 6792 DD1A 3853 | 983F FB9D 216F 36AC AAB3 3D26 DCE5 141B 5BE8 C802 F4F9 30A8 E8F3 5AD5 D135 53C3 |
| what is your name? | 21EB 497B 66EB 1A1A C2A6 3106 D6CF 3732 5C25 57AE | B7BC 3F43 8183 072B AF3E BB11 D3D6 3D78 1661 D4ED 7F1F 4D79 916F 89F5 1F56 17F7 |
| How old are you? | C0DC BA11 D4CF D9FF 2130 9597 CDFE D785 7868 7712 | 9B18 1DE4 83E0 DA61 391C D385 F163 83F9 5340 1FDF EE8C A2C4 6297 BD81 2F3E 0B5D |
| I'm a student | A44D 9889 2A49 3F27 7F9F 68CD 58D2 C5AF F9C3 6990 | F30F 4D24 6D07 2267 F9B8 9A5E B809 F26C 2EFA 0EE9 5F2B 7AB9 6BD2 E0B7 C4EB 3153 |
| Can you help me? | BF13 5557 FFF2 4CB8 BD58 BA66 50B8 A40C 2CEF 3B0E | F349 17CD C984 9D93 DC74 4DEA B27C 36C6 7262 119C B968 B227 9360 AF78 FF9A DF5D |
| Go straight | BF4A 182C 27DF AFEE AD4E C20B 458F 3476 4385 4FFF | 55BF D6F2 3246 2031 B00C 2892 3EE3 288F 1CE8 9163 7B21 6307 1400 4904 E78A 54FA |
| Happy new year | 9AC7 2AE1 B801 10EB 451E 2C4B CC4B 0010 AF22 921E | D2B8 5FA6 A231 E708 BD7D FA02 3A6E 6A51 C9DD B8EA F378 6E64 28B1 0B1B BC15 8DA5 |
| What do you do for a | 70ED AEF E 7730 5671 6A87 | E30F 7D7D 5B68 9D52 0F47 3749 47C0 02AF AC0D BE5C CC74 |





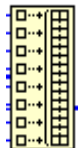


| | | |
|--|--|--|
| living? | 5E0C 473E 32B7 A67E FD10 | 1718 05E2 BE7B 2B5D 1B5C |
| I destroy my enemies when I make them my friends | 2761 4E0E D525 A1AB 1F48 9010 C7F0 D631 6B2B 76DB | 307E E207 F54C 3018 21B5 2A4F F3DB 4983 B460 F232 6EF3 F1A7 D376 02B3 3F9B 7BDB |
| If the plan doesn't work, change the plan, but never the goal | 2D33 F185 4D65 60DD 13A7 427C 2D6E 78E5 F051 FF75 | 9C3F ADB0 285A 3762 0BA5 3360 9AC7 403E A1F9 7D49 884A A8CF F3B8 FB22 5DE6 27A8 |
| Always keep your eyes open. Keep watching. Because whatever you see can inspire | 9EFC 13AC E333 852B 7DEF 6DC1 8F36 C0FB 4C3C 5570 | 95F4 6C28 7E1F A2AF C272 7AB3 B352 3CB9 7E08 7C6A 7185 1F8F 7DCE 48DF CF12 2B4F |






APPENDIX C





LABVIEW ICONS TABLE




| Icon symbol | Icon Name | Icon work | The aim of using in the AES and SHA design |
|---|-----------------------|---|--|
|  | Property Node | To select the class on which to execute the property | To access the private data of the AES |
|  | Invoke Node | To select the class on which to execute the method | Configuring the VI Server Application class or Virtual Instrument class |
|  | Constructor Node | Displays the Select .NET Constructor dialog box | This node identifies the constructor from which to create a .NET object. |
|  | Merge Errors Function | It has two inputs available and another inputs can be added. | To look for errors beginning with the error in 0 parameter and reports the first error found |
|  | Expression Node | Expression Nodes are useful when an expression has only one variable. | To calculate expressions that contain a single variable |
|  | String of characters | Generate a string of characters. | To make the string generated represent the original message to the system. |

| | | | |
|---|--|--|---|
|  | <p>String To Byte Array Function</p> | <p>Converts a string into an array of unsigned bytes. Each byte in the array has the ASCII value of the corresponding character in the string.</p> | <p>To convert the string which is the original message to its equivalent ASCII to be suitable for the next processing</p> |
|  | <p>Array Size Function</p> | <p>Returns the number of elements in each dimension of array.</p> | <p>Responsible of the block size, if the array is one dimension such as in the message string; then the returned value of this function is a 32-bit integer.</p> |
|  | <p>Quotient & Remainder Function</p> | <p>Computes the integer quotient and the remainder of the inputs.</p> | <p>Used in many places in algorithm, most of these to calculate the modulo value</p> |
|  | <p>Subtract Function</p> | <p>Computes the difference of the inputs.</p> | <p>Used in many places at the design to find the difference between two values.</p> |
|  | <p>Array Subset Function</p> | <p>Returns a portion of array starting at index and containing length elements</p> | <p>When an array wired to this function, the function resizes automatically to display index inputs for each dimension in the array. If the wired array is a 1 dimension array to the function, the function displays the index inputs for an element</p> |

| | | | |
|---|--|--|---|
|  | <p>To Unsigned Long Integer Function</p> | <p>Converts A Number To A 32-Bit Unsigned Integer In The Range 0 To $(2^{32})-1$</p> | <p>Used In Many Places, This Function Rounds All Floating-Point And Fixed-Point Numeric Values To The Nearest Integer. If The Fractional Part Of The Value Is .5, The Function Rounds The Value To The Nearest Even Integer. For Example, The Function Rounds 13.5 To 14 And Rounds 14.5 To 14.</p> |
|  | <p>Rotate Left With Carry Function</p> | <p>Rotates each bit in the input value one bit to the left (from least significant to most significant bit), inserts carry in the low-order bit, and returns the most significant bit.</p> | <p>Used in many places, especially at the padding and the core calculation to rotate the array when necessary.</p> |
|  | <p>Split Number Function</p> | <p>Breaks a number into its component bytes or words</p> | <p>Used at the padding process, to determine the MSB and LSB of the array after the rotating.</p> |
|  | <p>Reverse 1D Array Function</p> | <p>Reverses the order of the elements in array</p> | <p>Used at the padding after the splitting to reorder the array at the same initial sequence</p> |
|  | <p>Build Array Function</p> | <p>Concatenates multiple arrays or appends elements to an n-dimensional array. You also can use the Replace Array Subset function to modify an existing array</p> | <p>Used at the padding process to reorder each chunk entered to the process to take its place and to append the padded message with the reserved register</p> |
|  | <p>Decrement Function</p> | <p>Subtracts 1 from the input value</p> | <p>Used in many places at the design when needing to decrement the value.</p> |
|  | <p>Initialize Array</p> | <p>Creates an n-dimensional array in which every element is initialized to</p> | <p>Used to initial an array concludes the array followed by 1 and series</p> |

| | | | |
|---|-------------------------------|--|--|
| | Function | the value of element . Initialized array is an array of the same type as the type you wire to element . | of zeroes then followed by 64 bits to show the length before padding at one array. |
|  | Multiply Function | Returns the product of the inputs | Used in many places of algorithm. |
|  | Replace Array Subset Function | Replaces an element or subarray in an array at the point you specify in index | Used in many places at algorithms |
|  | Type Cast Function | Casts x to the data type, type , by flattening it and unflattening it using the new data type. If the function must reinterpret data instead of transforming it, LabVIEW uses a temporary buffer. | Used in many places of algorithm to convert to the suitable type, to dispose the error generated from data mismatching |
|  | Exclusive Or Function | Computes the logical exclusive or (XOR) of the inputs. Both inputs must be Boolean values, numeric values, or error clusters. If both inputs are TRUE or both inputs are FALSE, the function returns FALSE. Otherwise, it returns TRUE. | Used in many places of the SHA-1/SHA-2 core for necessary calculations. |
|  | Add Function | If two waveform values or two dynamic data type values wired to this function, error in and error out terminals appear on the function. You cannot add two time stamp values together. The dimensions of two matrices you want to add must be the same. Otherwise, this function returns an empty matrix. The connector pane displays the default data types for this polymorphic function | Used in many places of the algorithm inside the SHA-1 core calculations. |

| | | | |
|---|--|--|--|
|  | <p>Rotate Function</p> | <p>Rotates x the number of bits specified by y</p> | <p>Used in many places of the algorithm inside the SHA-1 core calculations</p> |
|  | <p>Replace Array Subset Function</p> | <p>Replaces an element or subarray in an array at the point you specify in index. When an array wired to this function, the function resizes automatically to display index inputs for each dimension in the array you wired. The connector</p> | <p>Used in many places</p> |
|  | <p>Compound Arithmetic Function</p> | <p>Performs arithmetic on one or more numeric, array, cluster, or Boolean inputs. To select the operation (Add, Multiply, AND, OR, or XOR), right-click the function and select Change Mode from the shortcut menu. When you select this function from the Numeric palette, the default mode is Add. When you select this function from the Boolean palette, the default mode is OR.</p> | <p>Used in many places</p> |
|  | <p>Number To Hexadecimal String Function</p> | <p>Converts number to a string of hexadecimal digits at least width characters wide or wider if necessary. The digits A–F always appear in uppercase in the output string. If number is floating-point or fixed-point, it is rounded to a 64-bit integer before conversion.</p> | <p>Used to display the hash code in Hex.</p> |

| | | | |
|---|---------------------------------|----------------------------------|---|
|  | <p>For Loop</p> | <p>SHA core</p> | <p>For repetition to the SHA rounds</p> |
|  | <p>Case structure</p> | <p>Case probability</p> | <p>For many cases</p> |
|  | <p>First Call? Function</p> | <p>Before the case structure</p> | <p>For proving the true and false cases</p> |

APPENDIX D

GALOIS FIELD (GF)

$GF(p^n)$ The case in which n is greater than one is much more difficult to describe. In cryptography, one almost always takes p to be 2 in this case. This section just treats the special case of $p = 2$ and $n = 8$, that is, $GF(2^8)$, because this is the field used by the new U.S. Advanced Encryption Standard (AES). The AES works primarily with bytes (8 bits), represented from the right as: $b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$. The 8-bit elements of the field are regarded as polynomials with coefficients in the field Z_2 :

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0x^0 \quad (D.1)$$

The field elements will be denoted by their sequence of bits, using two hex digits.

D.1 ADDITION IN $GF(2^8)$

To add two field elements, just add the corresponding polynomial coefficients using addition in Z_2 . Here addition is *modulo 2*, so that $1 + 1 = 0$, and addition, subtraction and exclusive-or are all the same. The identity element is just zero: 00000000 (in bits) or 0x00 (hex).

D.2 MULTIPLICATION IN $GF(2^8)$


Multiplication in this field is much more difficult and harder to understand, but it can be implemented very efficiently in hardware and software. The first step in multiplying two field elements is to multiply their corresponding polynomials just as in beginning algebra (except that the coefficients are only 0 or 1, and $1 + 1 = 0$ makes the calculation easier, since many terms just drop out). The result would be up to a degree 14 polynomial -- too big to fit into one byte. A finite field now makes use of a fixed degree eight irreducible polynomial (a polynomial that cannot be factored into the product of two simpler polynomials). For the AES the polynomial used is the following (other polynomials could have been used):

$$m(x) = x^8 + x^4 + x^3 + x + 1 = 0 * 11b \text{ (hex)} \quad (\text{D.2})$$

The intermediate product of the two polynomials must be divided by $m(x)$. The remainder from this division is the desired product. This sounds hard, but is easier to do by hand than it might seem (though error-prone). To make it easier to write the polynomials down, adopt the convention that instead of $x^8 + x^4 + x^3 + x + 1$ just write the exponents of each non-zero term. (Remember that terms are either zero or have a 1 as coefficient.).



CURRICULUM VITAE

| | | |
|---|---|--|
| <p>Name: (SARA ABDULHALEEM AL-SHAREA)</p> <p>M.Sc. (at 2019) in (TURKEY)</p> <p><u>Affiliation:</u> University of ALTINBAS</p> <p>Dept.: ELECTRIC AND COMPUTER ENGINEERING</p> <p>College: Graduate School of Social Science</p> <p>University: ALTINBAS</p> <p>Mobile: - +964- 7901414277 / +905396332236</p> <p>Specialization: - NETWORK SECURITY</p> <p><u>Your Research Interests:</u></p> <ul style="list-style-type: none"> - Cryptography - Computer networks - security <p>Google Site: Google scholar :</p> <p>https://scholar.google.com/citations?hl=ar&user=Xb-JkxsAAAAJ</p> <p>Research gate:</p> <p>https://www.researchgate.net/profile/Sara_Al-Sharea</p> <p>Academia.edu : sara.al-sharea@ogr.altinbas.edu.tr http://uzem.altinbas.edu.tr/user/profile.php?id=1834&lang=en</p> <p>LinkedIn: http://linkedin.com/in/sara-haleem-7a4ab27a</p> <p>Facebook: https://www.facebook.com/profile.php?id=100000258004945</p> | <p>sarahah_sarahen@yahoo.com sarahaleem26@gmail.com</p> <div style="text-align: center;">  </div> <p>-Are you a member in IEEE Iraq Section? If Yes, when you start your first registration? NO</p> <p>-Any Activities you have with IEEE Iraq Section???</p> <p>-Do you wish to be a reviewer in the IEEE conferences hold in IRAQ? YES</p> | <p>MY EADS ID: 1628116</p> |
|---|---|--|

