EFFICIENT HARDWARE IMPLEMENTATION FOR THE ADVANCED ENCRYPTION STANDARD AND RC6 ALGORITHM

Thesis report submitted in the partial fulfilment of the requirement for the award of the degree of

MASTER OF TECHNOLOGY

IN

VLSI DESIGN & CAD

Submitted by

Amandeep Kaur

Roll No.: 600961003

Under the Guidance of

Ms. Manu Bansal

Assistant Professor, ECED

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

THAPAR UNIVERSITY, PATIALA (PUNJAB) - 147004

JULY - 2011

CERTIFICATE

I hereby certify that the work which is being presented in this thesis entitled "**Efficient Hardware Implementations for the Advanced Encryption Standard and RC6 Algorithm**", is an authentic record of my own work carried out as the requirements for the award of degree of Master of Technology in VLSI Design & CAD at Thapar University, Patiala under the guidance of Ms. Manu Bansal, Assistant Professor, Electronics and Communication Engineering Department, Thapar University.

The matter presented in this thesis has not been submitted in any other university/Institute for the award of any degree.

Dated: 15. July. 2011

Amandeep Kaur

(Amandeep Kaur)

Roll No. 600961003

This is to certify that the above statement made by the candidate is correct and to the best of my knowledge.

Manu Bansal

(Ms. Manu Bansal)

Assistant Professor, ECED

Thapar University, Patiala

Counter Signed by:

A.K.Chatterji 20.7.11.

(Dr. A. K. Chatterjee)

Head, ECED

Thapar University, Patiala

(Dr. S. K. Mohapatra)

Dean of Academics Affairs

Thapar University, Patiala

ACKNOWLEDGEMENT

I wish to express my sincere gratitude to my supervisor **Ms. Manu Bansal** for her invaluable guidance and advice during every stage of this endeavour. I am greatly indebted to her for encouragement and support without which, it would not have been possible for me to complete this undertaking successfully. Her insightful comments and suggestions have continually helped me to improve my understanding.

My sincere thanks are due to **Dr. A.K. Chatterjee**, Head of the Department, Department of Electronics and Communication Engineering for providing the constant encouragement and providing the facilities in the department for the completion of my thesis work. I express my gratitude towards **Dr. Alpana Agarwal**, PG Coordinator, Department of Electronics and Communication Engineering**,** for her valuable guidance and encouragement.

I also want to thank my friends Puneet and Simran for accompanying me during the most outstanding year of my life and by me in every situation.

I take pride of myself being daughter of ideal parents for their over lasting desire, sacrifice, affection blessings and help without which it would not have been possible for me to complete my studies. I would like to thank my brothers and bhabhi for moral encouragement and support.

Last but not least, I would like to thank God for all good deeds.

(Amandeep Kaur)

ABSTRACT

Most cryptographic algorithms function more efficiently when implemented in hardware than in software running on single processor. However, systems that use hardware implementations have significant drawbacks: they are unable to respond to flaws discovered in the implemented algorithm or to changes in standards. As an alternative, it is possible to implement cryptographic algorithms in software running on multiple processors. However, most of the cryptographic algorithms like DES (Data Encryption Standard) or 3DES have some drawbacks when implemented in software: DES is no longer secure as computers get more powerful while 3DES is relatively sluggish in software. AES (Advanced Encryption Standard), which is rapidly being adopted worldwide, provides a better combination of performance and enhanced network security than DES or 3DES by being computationally more efficient than these earlier standards. Furthermore, by supporting large key sizes of 128, 192, and 256 bits, AES offers higher security against brute-force attacks.

In this thesis, AES and RC6 have been implemented. A brief overview and understanding of the RC6 Block Cipher algorithm is given. A comparison to the criteria and requirements given by the National Institute of Standards for consideration as the new Advanced Encryption Standard is made. So it is concluded that the RC6 algorithm is a good solution and applies well to most platforms, but not to all due to performance and security issues. The VHDL coding of AES algorithm and RC6 algorithm, their FPGA implementation by Xilinx Synthesis Tool on Virtex II pro kit have been done. The output of RC6 has been displayed on LCD of Spartan 3E kit. The synthesis results show that the computation time for generating the ciphertext by AES with 16 sbox and 2 dual port RAM is 4.403 ns, while for the architecture with 16 sbox and 8 dual port RAM is 5.463 ns and for architecture with 4 sbox and 2 dual port RAM is 6.922 ns.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Acronyms, Definitions and Symbols

This thesis document uses the following set of Acronyms, definitions and Symbols

AES:	Advanced Encryption Standard
Affine Transformation:	A mathematical operation of multiplication by a matrix followed by addition of a vector
Array:	A group of similar elements
Bit:	A binary value consisting of 1 or 0
Block:	Sequence of bits or array of bytes whose length is determined by the number of bits or bytes
Byte:	An array of 8 bits
Cipher:	Set of well-defined steps to transform data from plain text to cipher text
Cipher Key:	The secret key which is the password to encrypt the data. It also generates the set of keys for different iterations.
Cipher text:	The result of a cipher or input to inverse cipher
DES:	Data Encryption Standard
GF (p):	Finite field or Galois field over p which contains all the elements from 0 to (p-1)
Gbps:	Giga bits per second
FPGA:	Field programmable gate array
Plain text:	Input to the cipher
Rijndael:	Cryptographic algorithm specified in AES
Round Key:	The Key generated from Key Expansion
State:	Intermediate results of ciphers which are stored as a block which is an array of bytes or bits
S-box:	A nonlinear substitution table used for byte substitution in the Cipher and Key Expansion

Word:	An array of 4 Bytes or an array of 32 Bits
N_b:	Number of columns consisting of 32 bits which comprises a State. N_b =4 for this document
N_k:	Number of columns consisting of 32 bits of Cipher Key.
	N_k = 4 for this Document as this deals with only 128 bit key
N_r:	Number of rounds which are 10 for this document
XOR:	Exclusive or operations
•:	Finite field multiplication

CHAPTER 1

INTRODUCTION

1.1 Motivation

The emergence of the Internet as a trusted medium for commerce and communication has made cryptography an essential component of modern information systems. Cryptography provides the mechanisms necessary to implement accountability, accuracy and confidentiality in communications [1]. As demands for secure communication bandwidth grow, efficient cryptographic processing will become increasingly vital to good system performance. To introduce cryptography, an understanding of issues related to information security in general is necessary. Information security manifests itself in many ways according to the situation and requirement. Regardless of who is involved, to one degree or another, all parties to a transaction must have confidence that certain objectives associated with information security have been met.

Over the centuries, an elaborate set of protocols and mechanisms has been created to deal with information security issues when the information is conveyed by physical documents. Often the objectives of information security cannot solely be achieved through mathematical algorithms and protocols alone, but require procedural techniques and abidance of laws to achieve the desired result. For example, privacy of letters is provided by sealed envelopes delivered by an accepted mail service. The physical security of the envelope is, for practical necessity, limited and so laws are enacted which make it a criminal offense to open mail for which one is not authorized. It is sometimes the case that security is achieved not through the information itself but through the physical document recording it. For example, paper currency requires special inks and materials to prevent counterfeiting. Conceptually, the way information is recorded has not changed dramatically over time. Whereas information was typically stored and transmitted on paper, much of it now resides on magnetic media and is transmitted via telecommunications systems.

What has changed dramatically is the ability to copy and alter information. One can make thousands of identical copies of a piece of information stored electronically and

each is indistinguishable from the original. With information on paper, this is much more difficult. What is needed then for a society where information is mostly stored and transmitted in electronic form is a means to ensure information security which is independent of the physical medium recording or conveying it and such that the objectives of information security rely solely on digital information itself. One of the fundamental tools used in information security is the signature. It is a building block for many other services such as non-repudiation, data origin authentication, identification, and witnessing, to mention a few. Having learned the basics in writing, an individual is taught how to produce a handwritten signature for the purpose of identification.

At contract age the signature evolves to take on a very integral part of the person's identity. This signature is intended to be unique to the individual and serve as a means to identify, authorize, and validate. With electronic information the concept of a signature needs to be redressed; it cannot simply be something unique to the signer and independent of the information signed. Electronic replication of it is so simple that appending a signature to a document not signed by the originator of the signature is almost a triviality.

Achieving information security in an electronic society requires a vast array of technical and legal skills. There is, however, no guarantee that all of the information security objectives deemed necessary can be adequately met. The technical means is provided through cryptography.

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication [1]. Cryptography is not the only means of providing information security, but rather one set of techniques.

Cryptographic goals

The following four cryptographic goals form a framework from which other goals are derived:

1) **Confidentiality** is a service used to keep the content of information from all but those authorized to have it.
2) **Data integrity** is a service which addresses the unauthorized alteration of data. 3.
3) **Authentication** is a service related to identification.

4) **Non-repudiation** is a service which prevents an entity from denying previous commitments or actions. When disputes arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary.

A fundamental goal of cryptography is to adequately address these four areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities.

Cryptography, over the ages, has been an art practiced by many who have devised ad hoc techniques to meet some of the information security requirements. The last twenty years have been a period of transition as the discipline moved from an art to a science. There are now several international scientific conferences devoted exclusively to cryptography and also an international scientific organization, the International Association for Crypto logic Research (IACR), aimed at fostering research in the area.

1.2 Symmetric-key Cryptography and AES

Symmetric-key cryptography, also called secret key cryptography, is the most intuitive kind of cryptography. It involves the use of a secret key known only to the participants of the secure communication. Symmetric-key cryptography can be used to transmit information over an insecure channel, but it has also other uses, such as secure storage on insecure media or strong mutual authentication. In symmetric-key cryptography, the key must be shared by both the sender and the receiver. The sender applies the encryption function using the key to the plaintext to produce the ciphertext. The ciphertext is sent to the receiver, who then applies the decryption function using the same shared key. Since the plaintext cannot be derived from the ciphertext without knowledge of the key, the ciphertext can be sent over public networks such as the Internet. Therefore, symmetric key cryptography is characterized by the use of a single key to perform both the encrypting and decrypting of data. Since the algorithms are public knowledge, security is determined by the level of protection afforded the key. If key is kept secret, both the secrecy and authentication services are provided. Secrecy is provided, because if the message is intercepted, the intruder cannot transform the ciphertext into its plaintext format. Assuming that only two users know the key, authentication is provided because only a user with the key can generate ciphertext that a recipient can transform into meaningful plaintext.

The United States standard for Symmetric-key cryptography, in which the same key is used for both encryption and decryption, is the Data Encryption Standard (DES) [2]. This is based upon a combination and permutation of shifts and exclusive OR operations and so can be very fast when implemented directly on hardware (1 GByte/s throughput or better) or on general purpose processors. DES uses a 56-bit key and maps a 64-bit input block of plaintext onto a 64-bit output block of ciphertext. 56 bits is a rather small key for today's computing power, the key size is indeed one of the most controversial aspects of this algorithm. The mainstream cryptographic community has long held that DES's 56-bit key is too short to withstand a brute-force attack from modern computers [3]. The current key size of 56 bits (plus 8 parity bits) of DES is now starting to seem small, but the use of larger keys with triple DES (3DES) can generate much greater security. If security is the only consideration, then 3DES will be an appropriate choice for a standardized encryption algorithm for decades to come. However, the principle drawback of 3DES is that the algorithm is relatively sluggish in software. The original DES was designed for mid 1970s hardware implementation and does not produce efficient software code. 3DES, which has three times as many rounds as DES, is correspondingly slower. A secondary drawback is that both DES and 3DES use a 64- bit block size. For reasons of both efficiency and security, a larger block size is desirable. Because of these drawbacks, 3DES is not a reasonable candidate for long term use.

As a replacement, NIST (National Institute of Standards and Technology) of USA in 1997 issued a call for proposals for a new Advanced Encryption Standard (AES), which would have security length equal to or better than 3DES and significantly, improved efficiency. In addition to these general requirements, NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits. The AES algorithm was selected in October 2001 after a multi-year evaluation process led by NIST with submissions and review by an international community of cryptography experts and the Rijndael algorithm [4], invented by Joan Daemen and Vincent Rijmen, was selected as the standard which was published in November 2002. NIST's intent was to have a cipher that will remain secure well into the next century.

1.3 Aim of the Thesis

The main aims of this thesis are:

1) To implement AES on FPGA, find out its performance on single processor.
2) Compare the performance of various different styles of modelling for implementation of AES to find which implementation gives best performance.
3) Implementing RC6 as a candidate of AES.
4) Show which implementation of the AES offers better performance yet flexible enough for cryptographic algorithms.

1.4 Layout of the Thesis

This thesis is divided into six major parts. This chapter introduces the relationship between the information security and cryptography, cryptographic goals, concepts of symmetric-key cryptography, the most secured symmetric-key cryptographic algorithm: Advanced Encryption Standard (AES) and the aims of the thesis.

Chapter 2 gives a detailed discussion on symmetric–key and public-key cryptography, the importance of symmetric–key cryptography, limitations of DES and reason for using AES is discussed in this chapter.

Chapter 3 describes the overall structure and the working principle of AES.

Chapter 4 introduces the RC6 algorithm, its relation and comparison with AES.

Chapter 5 contains the brief introduction to Virtex 2pro and Spartan 3E kit.

Chapter 6 gives implementation steps on FPGA, power, area comparisons of various different styles of modelling of AES, their implementation with results and finally implementing RC6 algorithm, conclusion and future work.

CHAPTER 2

CONVENTIONAL ENCRYPTION AND AES

2.1 Introduction

Cryptography is generally understood to be the study of the principles and techniques by which information is converted into an encrypted version that is difficult (ideally impossible) for any unauthorized person to convert to the original information, while still allowing the intended reader to do so. In fact, cryptography covers rather more than merely encryption and decryption. It is, in practice, a specialized branch of information theory with substantial addition from other branches of mathematics. Cryptography is probably the most important aspect of communications security and is becoming increasingly important as a basic building block for computer security.

The increased use of computer and communications systems by the industry has increased the risk of theft of proprietary information. Although these threats may require a variety of countermeasures, cryptography is a primary method of protecting valuable electronic information. In data and telecommunications, cryptography is necessary when communicating over any unsecured medium, which includes just about any network, particularly the Internet. Within the context of any application-to-application communication, there are some specific security requirements, including:

Authentication*:* The process of proving one's identity.

Confidentiality*:* Ensuring that no one can read the message except the intended receiver.

Integrity*:* Assuring the receiver that the received message has not been altered in any way from the original.

Non-repudiation*:* A mechanism to prove that the sender really sent this message. There are, in general, two types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric or conventional) cryptography and public-key (or asymmetric) cryptography. In symmetric-key cryptography, an algorithm is used to scramble the message using a secret key in such a way that it becomes unusable to all except the ones that have

access to that secret key. The most widely known symmetric cryptographic algorithm is DES, developed by IBM in the seventies. It uses a key of 56 bits and operates on chunks of 64 bits at a time. In public key cryptography [4], algorithms use two different keys: a private and a public one. A message encrypted with a private key can be decrypted with its public key (and vice versa). The owner of the key pair holds the private key, and may distribute the public key to anyone. Someone who wants to send a secret message uses the public key of the intended receiver to encrypt it. Only the receiver holds the private key and can decrypt it.

The two basic building blocks of all encryption techniques are substitution and transposition. A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols. Transposition technique is a different kind of mapping where mapping is achieved by performing some sort of permutation on the plaintext letter.

In this chapter, symmetric–key and public-key cryptography will be discussed. Then the importance of symmetric–key cryptography, limitations of DES and reason for using AES (Advanced Encryption Standard) will be discussed.

2.2 Symmetric-key Cryptosystem

2.2.1 Principles of Symmetric-key Cryptosystem

In Symmetric-key cryptography, a single key is used for both encryption and decryption. The sender uses the key (or some set of rules) to encrypt the plaintext and sends the cipher text to the receiver. The receiver applies the same key (or rule set) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption. With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key.

Symmetric-key cryptography schemes are generally categorized as being either stream ciphers or block ciphers [5]. Stream ciphers operate on a single bit (byte or computer word) at a time, and implement some form of feedback mechanism so that the key is constantly changing. A block cipher is so-called because the scheme encrypts one block of data at a time using the same key on each block. In general, the same plaintext block will

always encrypt to the same cipher text when using the same key in a block cipher whereas the same plaintext will encrypt to different cipher text in a stream cipher.

Stream ciphers come in several flavours but two are widely used. Self-synchronizing stream ciphers calculate each bit in the key stream as a function of the previous *n* bits in the key stream. It is termed "self-synchronizing" because the decryption process can stay synchronized with the encryption process merely by knowing how far into the *n*-bit key stream it is. One problem is error propagation; a garbled bit in transmission will result in *n* garbled bits at the receiving side.

Synchronous stream ciphers generate the key stream in a fashion independent of the message stream but by using the same key stream generation function at sender and receiver. While stream ciphers do not propagate transmission errors, they are, by their nature, periodic so that the key stream will eventually repeat.

Block ciphers can operate in one of several modes; the following four are the most important:

Electronic Codebook (ECB) mode is the simplest, most obvious application: the secret key is used to encrypt the plaintext block to form a cipher text block. Two identical plaintext blocks, then, will always generate the same cipher text block. Although this is the most common mode of block ciphers, it is susceptible to a variety of brute-force attacks.

Cipher Block Chaining (CBC) mode adds a feedback mechanism to the encryption scheme. In CBC, the plaintext is exclusively-ORed (XORed) with the previous cipher text block prior to encryption. In this mode, two identical blocks of plaintext never encrypt to the same cipher text.

Cipher Feedback (CFB) mode is a block cipher implementation as a self-synchronizing stream cipher. CFB mode allows data to be encrypted in units smaller than the block size, which might be useful in some applications such as encrypting interactive terminal input. In case of 1-byte CFB mode, for example, each incoming character is placed into a shift register the same size as the block, encrypted, and the block transmitted. At the receiving side, the cipher text is decrypted and the extra bits in the block (i.e., everything above and beyond the one byte) are discarded.

Output Feedback (OFB) mode is a block cipher implementation conceptually similar to a synchronous stream cipher. OFB prevents the same plaintext block from generating the same cipher text block by using an internal feedback mechanism that is independent of both the plaintext and cipher text bit streams.

2.2.2 Model of Symmetric-key Cryptosystem

A symmetric or conventional encryption scheme has five ingredients [6] (Figure 2.1)**:**

1) **Plaintext/Message:** This is the original intelligible message or data that is fed into the algorithm as input.
2) **Encryption Algorithm:** The encryption algorithm performs various substitution and transformation on the plaintext.
3) **Secret Key:** The secret key is also the input to the encryption algorithm. The key is a value independent of the plaintext. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.
4) **Cipher text:** This is the scrambled message produced as output. It depends on the plaintext and secret key. For a given message, two different keys will produce two different cipher texts.
5) **Decryption Algorithm:** This is essentially the encryption algorithm run in reverse. It takes the cipher text and the secret key and produces the original plaintext.

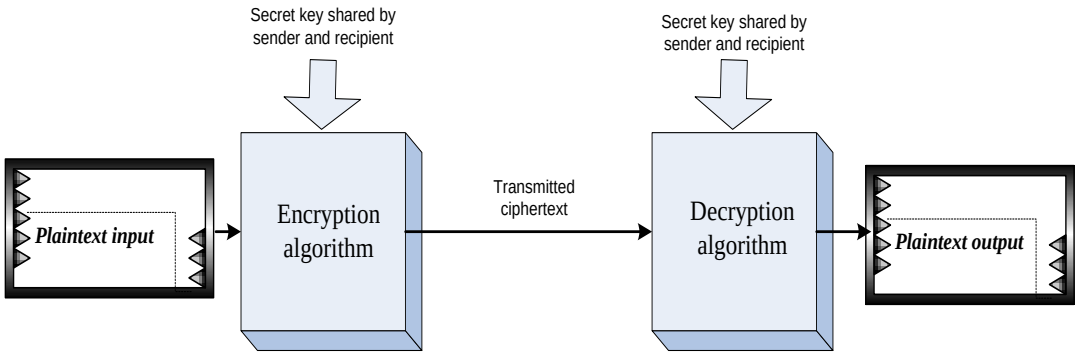


Figure 2.1 Model of Symmetric-key Cryptosystem

There are two requirements for secure use of symmetric-key encryption:

1) A strong encryption algorithm: At a minimum, the algorithm to be such that an opponent who knows the algorithm and has access to one or more cipher texts would be unable to decipher the cipher text or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt cipher text or discover the key even if he or she is in possession of a number of cipher texts together with the plaintext that produced each cipher text.
2) Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If some can discover the key and knows the algorithm, all communication using this key is readable.

In symmetric-key cryptosystem, a source produces a message in $X = [X_1, X_2, X_3, .. X_M]$ the plaintext, where M elements of X are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. The binary alphabet {0,1} is also typically used. For encryption, a key of the form $K = [K_1, K_2, K_3, \dots, K_J]$ is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination. With the message X and the encryption key K as input; the encryption algorithm forms the cipher text $Y = [Y_1, Y_2, Y_3, \dots .. Y_N]$. This process can be expressed using the following notation:

$$Y = E_k(X)$$

This notation indicates that Y is produced by using the encryption algorithm E as a function of the plaintext X, with the specific function determined by the value of the key. The intended receiver, in possession of the key, is able to invert the transformation:

$$X = E_k(Y)$$

2.3.3 Data Encryption Standard (DES)

The most common symmetric-key cryptography scheme used today is the Data Encryption Standard (DES), designed by IBM in the 1970s and adopted by the National Bureau of Standards (NBS) [now the National Institute for Standards and Technology (NIST)] in 1977 [2] for commercial and unclassified government applications. DES has been

adopted as Federal Information Processing Standard 46 (FIPS 46-3) and by the American National Standards Institute as X3.92. DES is a block-cipher employing a 56-bit key that operates on 64-bit blocks. DES has a complex set of rules and transformations that were designed specifically to yield fast hardware implementations and slow software implementations, although this latter point is becoming less significant today since the speed of computer processors is several orders of magnitude faster today than twenty years ago. IBM also proposed a 112-bit key for DES, which was rejected at the time by the government; the use of 112-bit keys was considered in the 1990s, however, conversion was never seriously considered.

Working Principle of DES:

DES uses a 56-bit key. In fact, the 56-bit key is divided into eight 7-bit blocks and an 8th odd parity bit is added to each block (i.e., a "0" or "1" is added to the block so that there is an odd number of 1 bits in each 8-bit block). By using the 8 parity bits for rudimentary error detection, a DES key is actually 64 bits in length for computational purposes (although it only has 56 bits worth of randomness, or entropy).

DES then acts on 64-bit blocks of the plaintext, invoking 16 rounds of permutations, swaps, and substitutes, as shown in Figure 2.2. The standard includes tables describing all of the selection, permutation, and expansion operations mentioned below; these aspects of the algorithm are not secrets. The basic DES steps are:

1) The 64-bit block to be encrypted undergoes an initial permutation (IP), where each bit is moved to a new bit position; e.g., the 1st, 2nd, and 3rd bits are moved to the 58th, 50th, and 42nd position, respectively.
2) The 64-bit permuted input is divided into two 32-bit blocks, called *left* and *right*, respectively. The initial values of the left and right blocks are denoted L0 and R0.
3) There are then 16 rounds of operation on the L and R blocks. During each iteration(where *n* ranges from 1 to 16), the following formulae apply:

$$L_n = R_n - 1$$

$$R_n = L_{n-1} XOR\, f(R_{n-1}, K_n)$$

At any given step in the process, the new L block value is merely taken from the prior R block value. The new R block is calculated by taking the bit-by-bit exclusive-OR (XOR)

of the prior L block with the results of applying the DES cipher function, *f*, to the prior R block and Kn. (Kn is a 48-bit value derived from the 64-bit DES key. Each round uses a different 48 bits according to the standard's Key Schedule algorithm.)

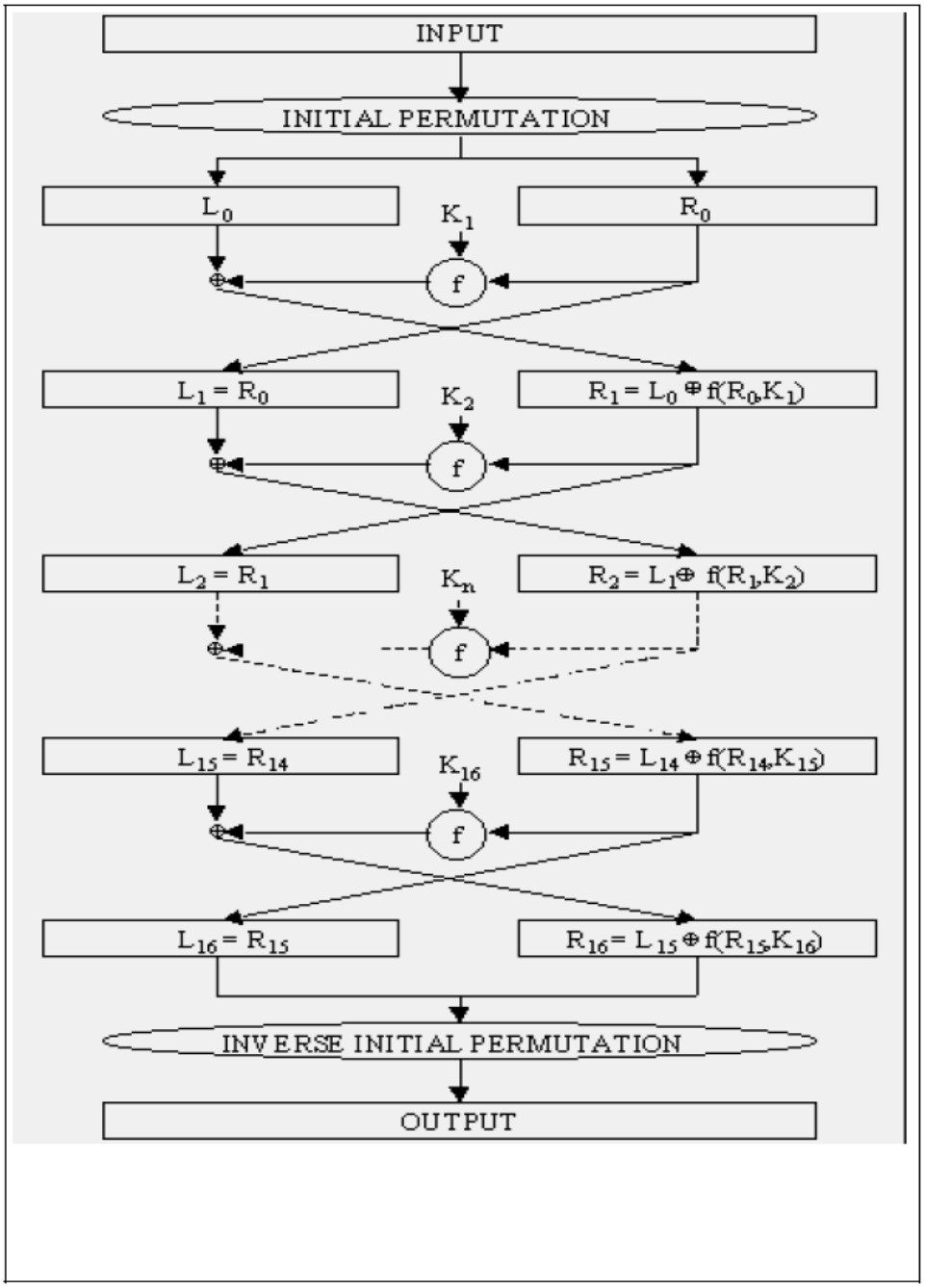


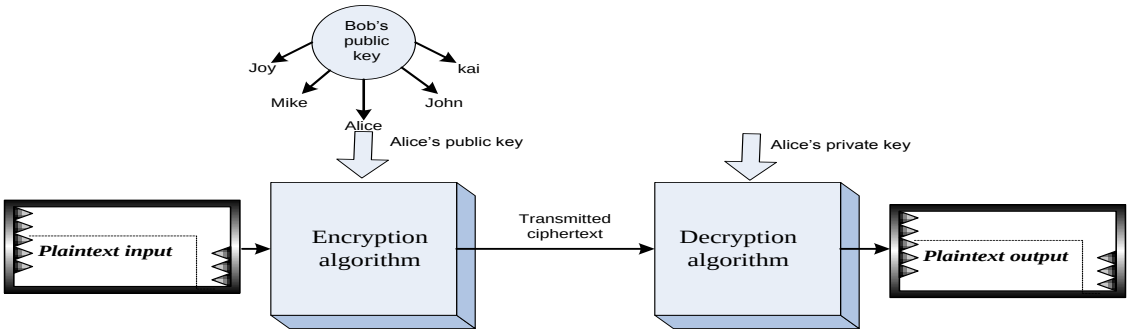
Figure 2.2 General depiction of DES encryption algorithm [14]

The cipher function, f, combines the 32-bit R block value and the 48-bit sub key in the following way. First, the 32 bits in the R block are expanded to 48 bits by an expansion function (E); the extra 16 bits are found by repeating the bits in 16 predefined positions. The 48-bit expanded R-block is then XORed with the 48-bit subkey. The result is a 48-bit value that is then divided into eight 6-bit blocks. These are fed as input into 8 selection (S) boxes, denoted S_1,..., S_8. Each 6-bit input yields a 4-bit output using a table lookup based on the 64 possible inputs; this results in a 32-bit output from the S-box. The 32 bits are then rearranged by a permutation function (P), producing the results from the cipher function.

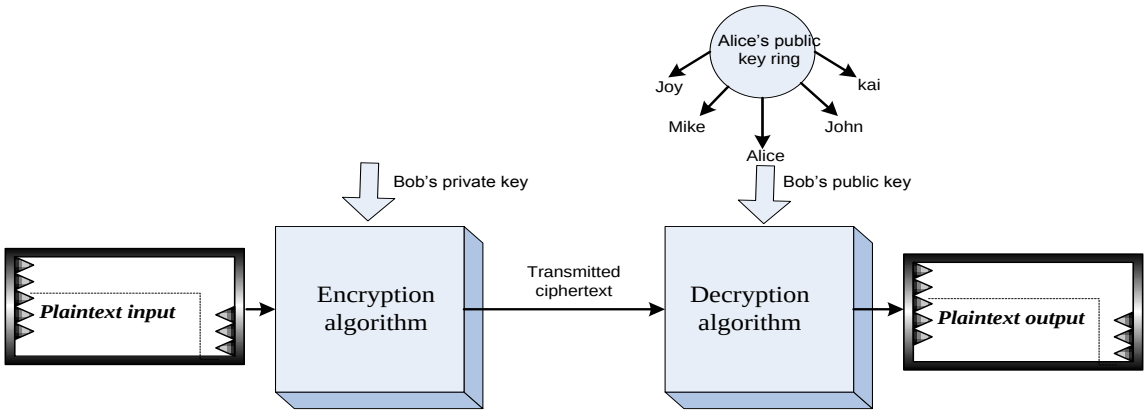
4) The results from the final DES round — i.e., L16 and R16 — are recombined into a 64-bit value and fed into an inverse initial permutation (IP-1). At this step, the bits are rearranged into their original positions, so that the 58th, 50th, and 42nd bits, for example, are moved back into the 1st, 2nd, and 3rd positions, respectively. The output from IP-1 is the 64-bit cipher text block.

2.3 Public Key Cryptography

Symmetric-key cryptography is based on the sender and receiver of a message knowing and using the same secret key: the sender uses the secret key to encrypt the message, and the receiver uses the same secret key to decrypt the message. The main problem is getting the sender and receiver to agree on the secret key without anyone else finding out. If they are in separate physical locations, they must trust a courier, or a phone system, or some other transmission medium to prevent the disclosure of the secret key being communicated.



(a) Encryption



(b) Authentication

Figure 2.3 Public-key Cryptography

2.3.1 RSA Algorithm

The first, and still most common, Public-key Cryptosystem implementation is RSA[9], named for the three MIT mathematicians who developed it - Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977 [7]. The basic technique was first discovered in 1973 by Clifford Cocks of CESG (part of the British GCHQ) but this was a secret until 1997. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers. The RSA scheme is a block cipher in which the plaintext and cipher text are integers between 0 and n-1 for some n. A typical size of n is 1024 bits or 309 decimal digits.

Plaintext is encrypted in blocks, with each block having a binary value less than some number n. That is, the block size must be less than or equal to $log_2(n)$; in practice, the block size is k bits, where $2k < n \leq 2k + 1$. Encryption and decryption are of the following form, for some plaintext block M and cipher text block C:

$$C = M^e mod\ n$$

$$M = C^d mod\ n = M^{ed} mod\ n$$

Both sender and receiver must know the value of n. The sender knows the value of e, and only the receiver knows the value of d. Thus this is a public-key encryption algorithm with a public-key of $K_u = \{e, n\}$ and a private Key of $K_R = \{d, n\}$.

For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:

- It is possible to find values of e, d, n such that $M^{ed} = M \ mod \ n$ for all M < n.
- It is relatively easy to calculate M^e and C^d for all values of M < n.
- It is infeasible to determine d given e and n.

Each participant must generate a pair of keys. This involves the following tasks:

- Selecting two prime numbers p and q.
- Calculating $n = pq$.
- Calculating $\phi(n) = (p-1)(q-1)$.
- Selecting e such that e is relatively prime to $\phi(n)$ and less than $\phi(n)$.
- Determining d such that $de = 1 \ mod \ \phi(n)$ and $d < \phi(n)$.

2.4 Importance of Symmetric-key Cryptography

The primary advantage of public-key cryptography is increased security and convenience. Private keys never need to transmitted or revealed to anyone. In a symmetric-key system, by contrast, the symmetric keys must be transmitted (either manually or through a communication channel), and there may be a chance that an enemy can discover the symmetric keys during their transmission.

Another major advantage of public-key systems is that they can provide a method for digital signatures. Authentication via symmetric-key systems requires the sharing of some symmetric keys and sometimes requires trust of a third party as well. As a result, a sender can repudiate a previously authenticated message by claiming that the shared symmetric key was somehow compromised by one of the parties sharing the symmetric-key.

Public-key authentication, on the other hand, prevents this type of repudiation; each user has sole responsibility for protecting his or her private key. This property of public-key authentication is often called non-repudiation.

A disadvantage of using public-key cryptography for encryption is speed; there are popular symmetric-key encryption methods that are significantly faster than any currently available public-key encryption method. Nevertheless, public-key cryptography can be used

with symmetric-key cryptography to get the best of both worlds. For encryption, the best solution is to combine public and symmetric-key systems in order to get both the security advantages of public-key systems and the speed advantages of symmetric-key systems. The public-key system can be used to encrypt a symmetric- key which is used to encrypt the bulk of a file or message. Such a protocol is called a digital envelope.

In some situations, public-key cryptography is not necessary and symmetric-key cryptography alone is sufficient. This includes environments where secure symmetric key agreement can take place, for example by users meeting in private. It also includes environments where a single authority knows and manages all the keys (e.g. closed banking system). Since the authority knows everyone's keys already, there is not much advantage for some to be "public" and others "private." Also, public-key cryptography is usually not necessary in a single-user environment. In general, public-key cryptography is best suited for an open multi-user environment.

Public-key cryptography is not meant to replace symmetric-key cryptography, but rather to supplement it, to make it more secure. The first use of public-key techniques was for secure key exchange in an otherwise symmetric-key system; this is still one of its primary functions. Symmetric-key cryptography remains extremely important and is the subject of on-going study and research.

Advantages of symmetric-key cryptography

1) Symmetric-key ciphers can be designed to have high rates of data throughput.
2) Keys for symmetric-key ciphers are relatively short.
3) Symmetric-key ciphers can be employed as primitives to construct various cryptographic mechanisms including pseudorandom number generators, hash functions and computationally efficient digital signature schemes, to name just a few.
4) Symmetric-key ciphers can be composed to produce stronger ciphers. Simple transformations which are easy to analyse, but on their own weak, can be used to construct strong product ciphers.

Disadvantages of symmetric-key cryptography

1) In a two-party communication, the key must remain secret at both ends.
2) In a large network, there are many key pairs to be managed. Consequently, effective key management requires the use of an unconditionally trusted TTP.
3) In a two-party communication between entities A and B, sound cryptographic practice dictates that the key be changed frequently and perhaps for each communication session.
4) Digital signature mechanisms arising from symmetric-key encryption typically require either large keys for the public verification function or the use of a TTP.

Advantages of public-key cryptography

1) Only the private key must be kept secret (authenticity of public keys must, however, be guaranteed).
2) Depending on the mode of usage, a private key/public key pair may remain unchanged for considerable periods of time, e.g., many sessions (even several years).
3) Many public-key schemes yield relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart.
4) In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario.

Disadvantages of public-key encryption

1) Throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than the best-known symmetric-key schemes.
2) Key sizes are typically much larger than those required for symmetric-key encryption, and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.

Summary of comparison

1) Public-key cryptography facilitates efficient signatures (particularly nonrepudiation) and key management, and

2) Symmetric-key cryptography is efficient for encryption and some data integrity applications.

2.5 Limitations of DES

In cryptography, key size does matter. The larger the key, the harder it is to crack a block of encrypted data. The reason that large keys offer more protection is almost obvious; computers have made it easier to attack cipher text by using brute force methods rather than by attacking the mathematics. With a brute force attack, the attacker merely generates every possible key and applies it to the cipher text. Any resulting plaintext that makes sense offers a candidate for a legitimate key.

Until the mid-1990s or so, brute force attacks were beyond the capabilities of computers that were within the budget of the attacker community. Today, however, significant compute power is commonly available and accessible. General purpose computers such as PCs are already being used for brute force attacks. For serious attackers with money to spend, such as some large companies or governments, Field Programmable Gate Array (FPGA) or Application-Specific Integrated Circuits (ASIC) technology offers the ability to build specialized chips that can provide even faster and cheaper solutions than a PC. An AT & T ORCA chip (FPGA) costs $200 and can test 30 million DES keys per second, while a $10 ASIC chip can test 200 million DES keys per second (compared to a PC which might be able to test 40,000 keys per second).

The mainstream cryptographic community has long held that DES's 56-bit key was too short to withstand a brute-force attack from modern computers. DES is even more vulnerable to a brute-force attack because it is often used to encrypt words, meaning that the entropy of the 64-bit block is, effectively, greatly reduced. That is, if encryption is performed on random bit streams, then a given byte might contain any one of 2^8 (256) possible values and the entire 64-bit block has 264, or about 18.5 quintillion, possible values.

If encryption is performed on words, however, it is most likely to find a limited set of bit patterns; perhaps 70 or so if the upper and lower case letters, the numbers, space, and some punctuation are considered. This means that only about ¼ of the bit combinations of a given byte are likely to occur. In mid 1990, RSA Laboratories sponsored a series of cryptographic challenges to prove that DES was no longer appropriate for use.

DES Challenge I was launched in March 1997. It was completed in 84 days by R. Verser in a collaborative effort using thousands of computers on the Internet.

The first DES II challenge lasted 40 days in early 1998. This problem was solved by distributed net, a worldwide distributed computing network using the spare CPU cycles of computers around the Internet (participants in distributed. net's activities load a client program that runs in the background. The distributed.net systems were checking 28 billion keys per second by the end of the project.

The second DES II challenge lasted less than 3 days. On July 17, 1998, the Electronic Frontier Foundation (EFF) announced the construction of hardware that could brute force a DES key in an average of 4.5 days. Called Deep Crack, the device could check 90 billion keys per second and cost only about $220,000 including. Since the design is scalable, this suggests that an organization could build a DES cracker that could break 56-bit keys in an average of a day for as little as $1,000,000.

The DES III challenge, launched in January 1999, was broken is less than a day by the combined efforts of Deep Crack and distributed.net. This is widely considered to have been the final nail in DES's coffin.

2.6 AES: An Alternative to DES

From the above discussion, it is obvious that the symmetric-key cryptography is efficient for encryption while the Public-key cryptography facilitates efficient signatures (particularly non-repudiation) and key management.

Symmetric–key cryptography is faster than any currently available public-key encryption method. On the other hand, the most widely used symmetric-key encryption technique like DES is vulnerable to a brute-force attack because of its inadequate key size compare to the processing power of modern computer. In order to increase the security of symmetric-key cryptography, NIST in 1997 issued a call for proposals for a new Advanced Encryption Standard (AES), which should have security strength better than DES and significantly improved efficiency. In addition, to these general requirements, NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits. Chapter 3 presents working principle of AES.

CHAPTER 3

WORKING PRINCIPLE OF AES

The search for a replacement to DES started in January 1997 when NIST (National Institute of Standards and Technology) of the USA announced that it was looking for an Advanced Encryption Standard [6]. In September of that year, they put out a formal Call for Algorithms and in August 1998 [2] announced that 15 candidate algorithms were being considered (Round 1). In April 1999 [11], NIST announced that the15 had been whittled down to five finalists (Round 2): MARS (multiplication, addition, rotation and substitution) from IBM; Ronald Rivest'sRC6; Rijndael from a Belgian team; Serpent, developed jointly by a team from England, Israel, and Norway; and Twofish, developed by Bruce Schneier. In October 2000, NIST announced their selection: Rijndael [13].

3.1 The AES Cipher

The Rijndael proposal for AES [8] defined a cipher in which the block length and the key length specified to be 128, 192, or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. A number of AES parameters (Table 3.1) depend on the key length. Most of the implementation of AES uses the key length of 128 bits.

Table 3.1 AES parameter

Key size (words/byte/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size(word/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

Rijndael was designed to have the following characteristics:

• Resistance against all known attacks.

• Speed and code compactness on a wide range of platforms.

• Design simplicity.

3.2 Overall Structure of AES

The overall structure of AES is depicted in figure 3.1. The input to the encryption and decryption algorithms is a single 128-bit block. This block of input is depicted as a square matrix of bytes. This block is copied into the state array, which is modified at each stage of encryption or decryption. After the final stage, state is copied to an output matrix. These operations are depicted in figure 3.2. Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words; each word is four bytes and total key schedule is 44 words for the 128-bit key. The ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the in matrix, the second four bytes occupy the second column and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the w matrix.

Several features of the overall AES structure [8]:

1) One noteworthy feature of this structure is that it is not a Feistel structure. In the classic Feistel structure, half of the data block is used to modify the other half of the data block, and then the half are swapped. Rijndael does not use a Feistel structure but process the entire block in parallel during each round using substitutions and permutations.
2) The key that is provided as input is expanded into an array of forty-four 32-bit words, w[i]. Four distinct words (128 bits) serve as a round key for each round; these are indicated in Figure 3.1.
3) Four different stages are used, one of permutation and three of substitution:
   I. **Substitute bytes**: Uses an S-box to perform a byte-by-byte substitution of the block
   II. **Shift rows**: A simple permutation
   III. **Mix Columns:** A substitution that makes use of arithmetic over GF (2^8)
   IV. **Add round Key:** A simple bitwise XOR of the current block with a portion of the expanded key
4) The structure of AES is quite simple. For both encryption and decryption, the cipher begins with an Add Round Key stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Figure 3.3 depicts the structure of a full encryption round.

5) Only the Add Round Key stage makes use of the key. For this reason, the cipher begins and ends with an Add Round Key stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
6) The Add Round Key stage is, in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key. The cipher is an alternating operations of XOR encryption (Add Round Key) of a block, followed by scrambling of the block (the other three stages), and followed by XOR encryption and so on. This scheme is both efficient and highly secure.

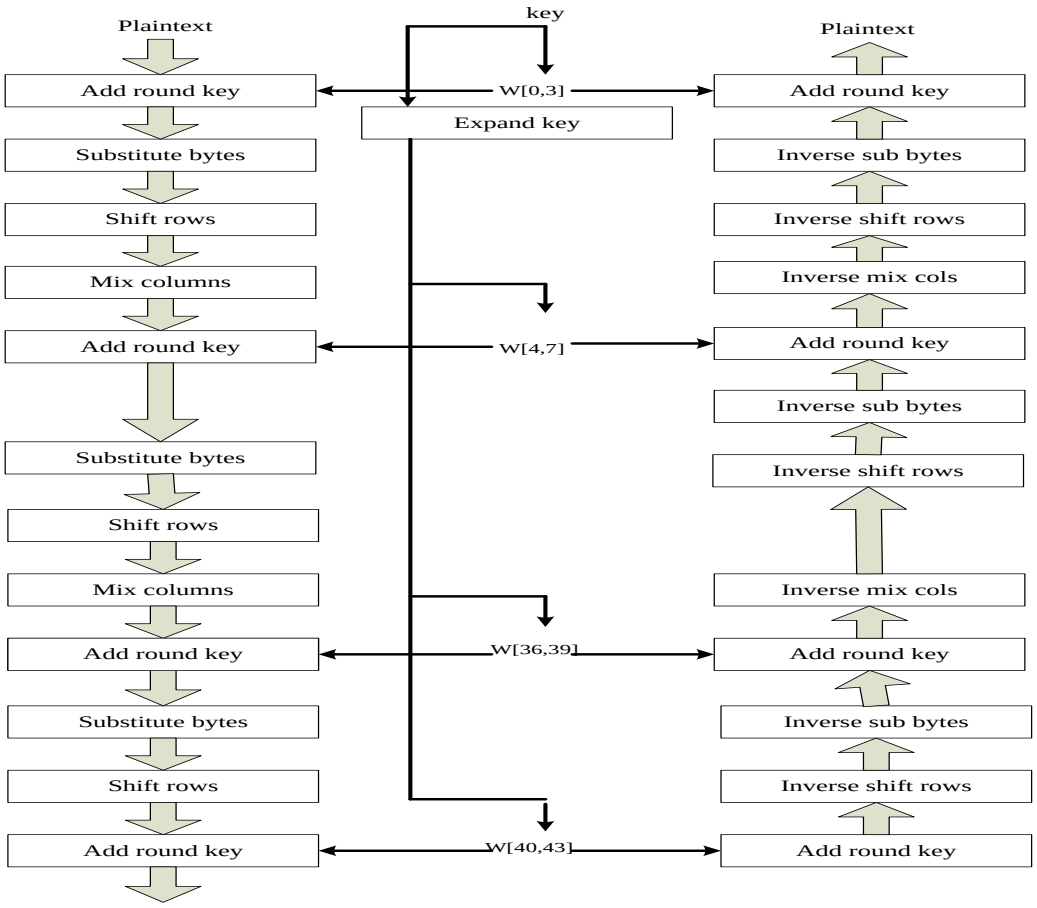


Figure 3.1 AES encryption and decryption

7) Each stage is easily reversible. For the Substitute Byte, Shift Row, and Mix Columns stages, an inverse function is used in the decryption algorithm. For the Add Round Key stage, the inverse is achieved by XORing the same round key to the block, using the result that A ⊕A⊕B = B.

8) As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.

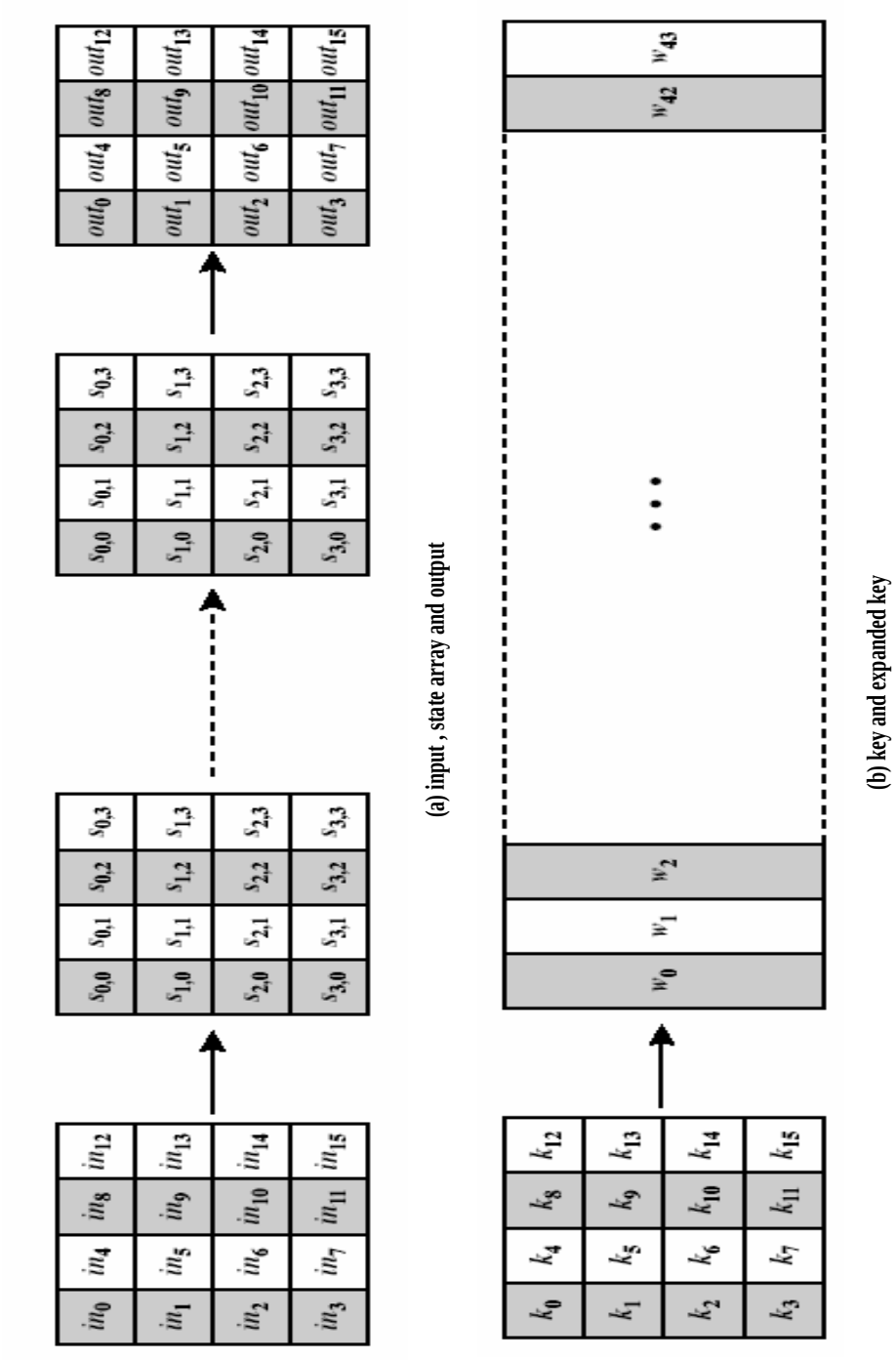


Figure 3.2 AES data structures [24]

9) Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure 3.1 lays out encryption and decryption

going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), State is the same for both encryption and decryption.

10) The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

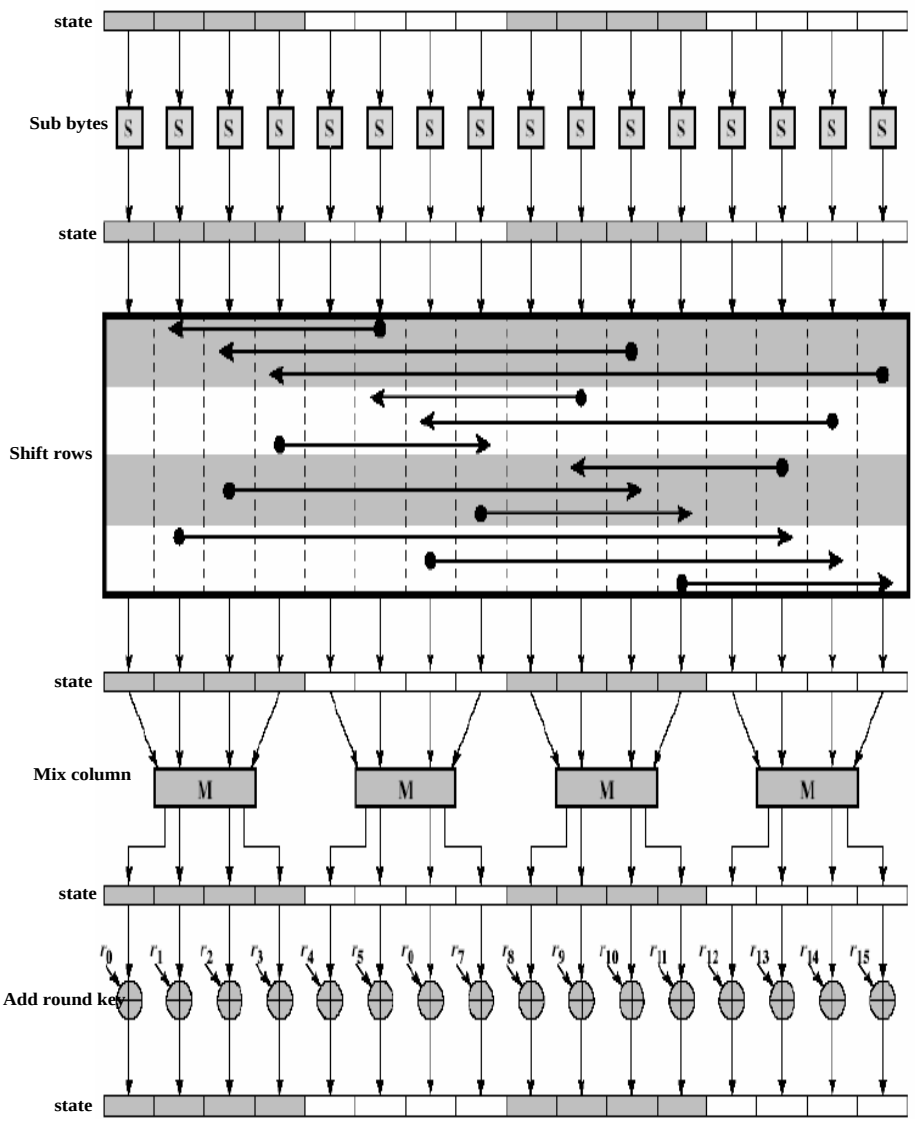


Figure 3.3 AES encryption round [24]

Now, all the four stages that are used in AES will be discussed. For each stage, the forward (encryption) algorithm, the inverse (decryption) algorithm, and the rational for the stage will be described. This is followed by a discussion of key expansion.

3.3 Substitute Bytes Transformation

The forward substitute byte transformation**,** called Sub Bytes [12], is a simple table lookup (Figure 3.4a). AES defines a 16×16 matrix of byte values, called a S-box (Table 3.2a), that contains a permutation of all possible 256 8-bit values. Each individual byte of State is mapped into a new byte in the following way:

The leftmost4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

The S-box is constructed in the following fashion:

1) Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02}, {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row *x*, column *y* is {*xy}*.
2) Map each byte in the S-box to its multiplicative inverse in the finite field GF (2^8); the value {00} is mapped to itself.
3) Each byte in the S-box consists of 8 bits labelled *(b7, b6, b5, b4, b3, h2, b1,b0).*

The following transformation is applied to each bit of each byte in the Sbox:

$$b_i' = b_i \oplus b_{(i+4)} \oplus mod\ 8 \oplus b_{(i+5)}\ mod\ 8 \oplus b_{(i+6)} mod\ 8 \oplus b_{(i+7)}\ mod\ 8 \oplus C_i \quad 3.1$$

where C_i is the i^{th} bit of byte C with the value {63},i.e.

(C7C6CSC4C3C2C1C0) = (01100011).

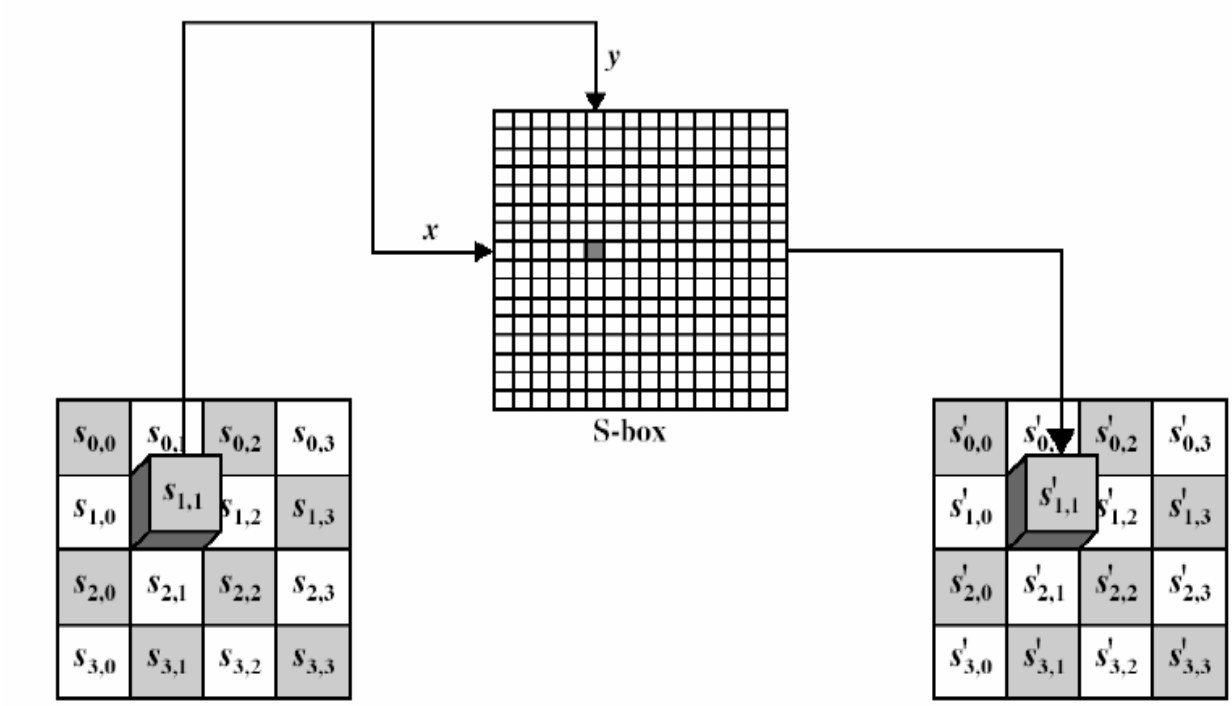
The prime (') indicates that the variable is to be updated by the value on the right. The AES standard depicts this transformation in matrix form as follows:

In ordinary matrix multiplication, each element in the product matrix is the sum of products of the elements or one row and one column.

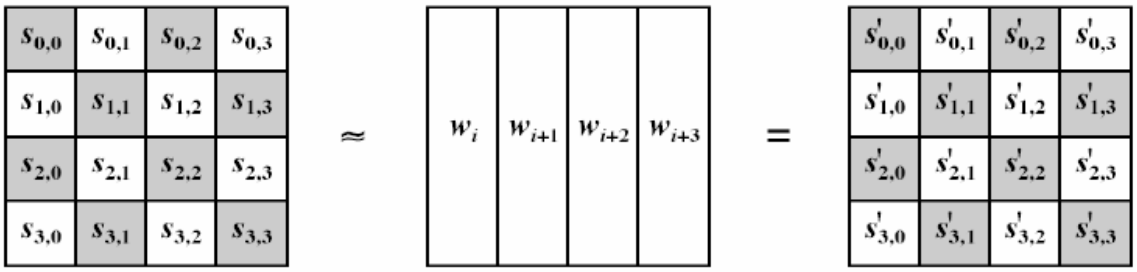
In this case, each element in the product matrix is the bitwise XOR of products of elements of one row and one column. Further, the final addition shown in the Equation 3.2 is a bitwise XOR.

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 10001111 \\ 11000111 \\ 11100011 \\ 11110001 \\ 11111000 \\ 01111100 \\ 00111110 \\ 00011111 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_0 \\ b_0 \\ b_0 \\ b_0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

3.2



(a) Substitute byte transformation



(b) Add round key transformation

Figure 3.4 AES byte-level operations

Table 3.2 AES S-boxes [12]

(a) S-box

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

(b) Inverse S-box

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

As an example, the input value {95} is considered. The multiplicative inverse in GF (2^8) is $\{95\}^{-1}$ = (8A), which is 10001010 in binary. Using the above Equation,

$$\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\end{bmatrix}\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}\oplus\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}=\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}\oplus\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}=\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}$$

The result is (2A), which will appear in row {09} column {05} of the S-box. This is verified by checking Table 3.2a.

The inverse substitute byte transformation, called InvSubBytes, makes use of the inverse S-box shown in Table 3.2b. The input {2AJ produces the output {95J, and the input (95J to the S-box produces {2A}. The inverse S-box is constructed by applying the inverse of the transformation in Equation (3.1) followed by taking the multiplicative inverse in GF (2^8). The inverse transformation is:

$$b_i^{'} = b_i \oplus b_{(i+2)} \oplus mod\ 8 \oplus b_{(i+5)}\ mod\ 8 \oplus b_{(i+7)} mod\ 8 \oplus d_i \qquad 3.3$$

Where byte d= {05}, or 00000101.

It can be represented as follows:

$$\begin{bmatrix}b'_0\\b'_1\\b'_2\\b'_3\\b'_4\\b'_5\\b'_6\\b'_7\end{bmatrix}=\begin{bmatrix}0&0&1&0&0&1&0&1\\1&0&0&1&0&0&1&0\\0&1&0&0&1&0&0&1\\1&0&1&0&0&1&0&0\\0&1&0&1&0&0&1&0\\0&0&1&0&1&0&0&1\\1&0&0&1&0&1&0&0\\0&1&0&0&1&0&1&0\end{bmatrix}\begin{bmatrix}b_0\\b_1\\b_2\\b_3\\b_4\\b_5\\b_6\\b_7\end{bmatrix}+\begin{bmatrix}1\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}$$

To verify that InvSubBytes is the inverse of SubBytes, the matrices in SubBytesand InvSubBytes are labelled as X and Y, respectively, and the vector versions of constants c and d are labelled as C and D, respectively. For some 8-bit vector B, Equation (3.2) becomes B' = XB ⊕C. It must be proved that Y (XB ⊕C) ⊕D = B multiply out, It must satisfy that YXB ⊕YC ⊕D = B. This becomes

$$\begin{bmatrix} 0&0&1&0&0&1&0&1 \\ 1&0&0&1&0&0&1&0 \\ 0&1&0&0&1&0&0&1 \\ 1&0&1&0&0&1&0&0 \\ 0&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&1 \\ 1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \end{bmatrix} \begin{bmatrix} 1&0&0&0&1&1&1&1 \\ 1&1&0&0&0&1&1&1 \\ 1&1&1&0&0&0&1&1 \\ 1&1&1&1&0&0&0&1 \\ 1&1&1&1&1&0&0&0 \\ 0&1&1&1&1&1&0&0 \\ 0&0&1&1&1&1&1&0 \\ 0&0&0&1&1&1&1&1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus$$

$$\begin{bmatrix} 0&0&1&0&0&1&0&1 \\ 1&0&0&1&0&0&1&0 \\ 0&1&0&0&1&0&0&1 \\ 1&0&1&0&0&1&0&0 \\ 0&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&1 \\ 1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

It is proved from the above equation that YX equals to the identity matrix, and the YC = D, so that YC ⊕D equals the null vector.

The S-box is designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design that has a low correlation between input bits and outputs bits, and the property that the output cannot be described as a simple mathematical function of the input. In addition, the constant in Equation (3.1) was chosen so that the S-box has no fixed points [S-box (a) = a] and no "opposite fixed points" [S=box (a) = ā], where ā is the bitwise complement of a.

The S-box must be invertible, that is, IS-box[S-box (a)] =a. However, the S-box is not self-inverse in the sense that it is not true that S-box (a) =IS-box (a). For example, Sbox ({95}) = {2A}, but IS-box ({95}) = {AD}.

3.4 Shift Row Transformation

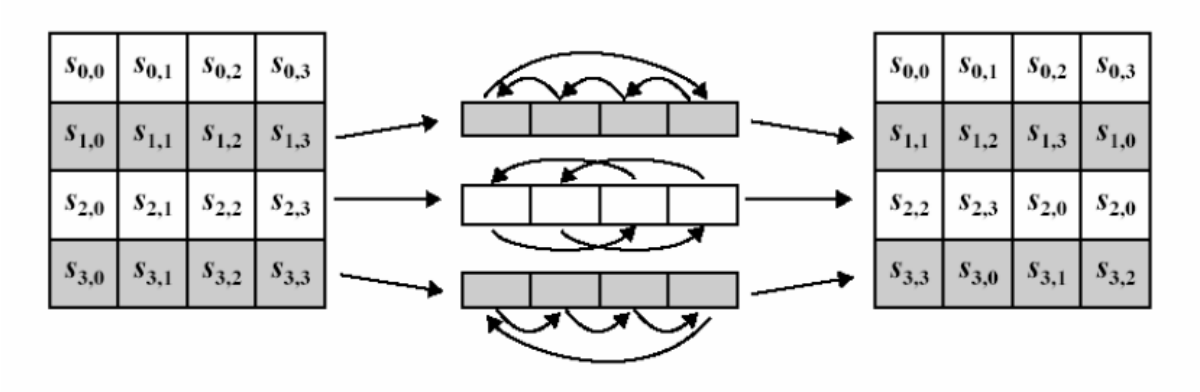


Figure 3.5 Shift row transformations [17]

The forward shift row transformation, called ShiftRows [11], is depicted in Figure 3.5. The first row of state is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following is an example of shiftRows:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

The inverse shift row transformation, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a one-byte circular right shift for the second row, and so on.

The shift row transformation is more substantial than it may first appear. This is because the State, as well as the cipher input and output, is treated as an array of four 4-byte columns. Thus, on encryption, the first 4 bytes of the plaintext are copied to the first column of State, and so on. However, the round key is applied to State column by column. Thus, a row shift moves an individual byte from one column to another, which is

a linear distance of a multiple of 4 bytes. Moreover, the transformation ensures that the 4 bytes of one column are spread out to four different columns.

3.5 Mix Column Transformation

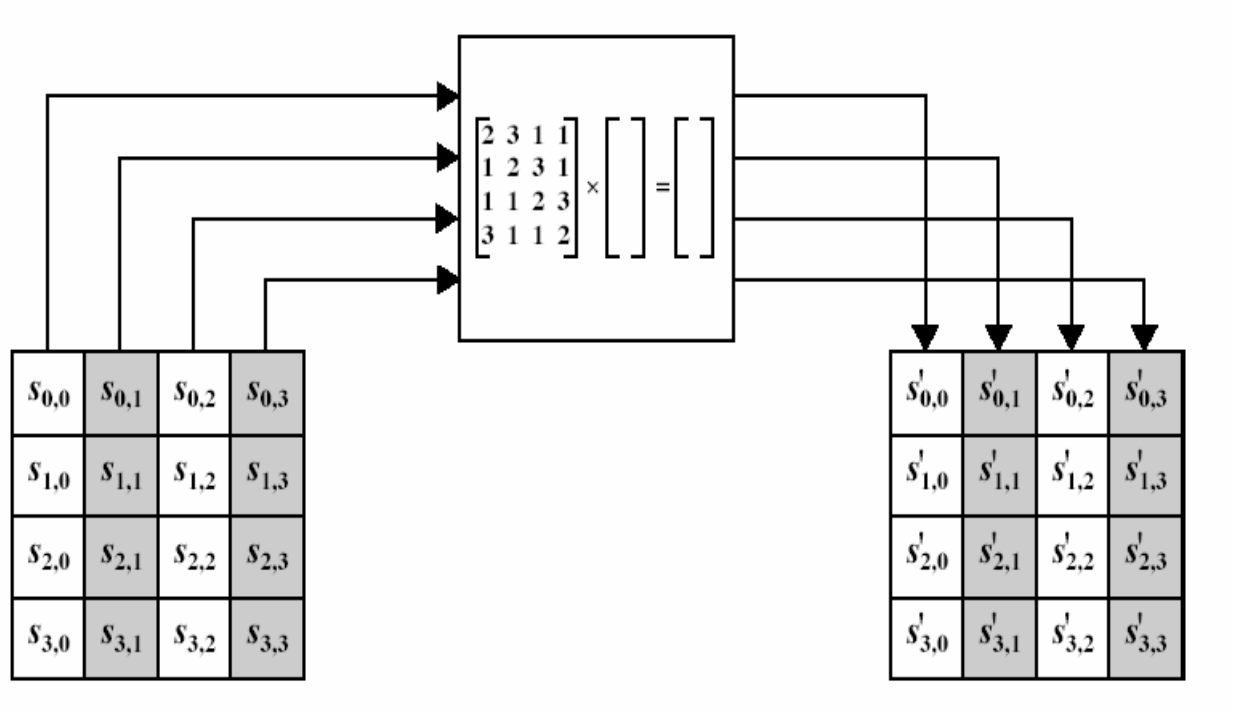


Figure 3.6 Mix column transformations

The forward mix column transformation, called Mix Columns [12], operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in the column. The transformation can be defined by the following matrix multiplication on State.

$$\begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix} \tag{3.4}$$

Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in GF (2^8).

The MixColumns transformation on a single column j ($0 \leq j \leq 3$) of State can be expressed as:

$$S'_{0,j} = (2 \bullet S_{0,j}) \oplus (3 \bullet S_{1,j}) \oplus S_{2,j} \oplus S_{3,j}$$

$$S'_{1,j} = S_{0,j} \oplus (2 \bullet S_{1,j}) \oplus (3 \bullet S_{2,j}) \oplus S_{3,j}$$

$$S'_{2,j} = S_{0,j} \oplus S_{1,j} \oplus (2 \bullet S_{2,j}) \oplus (3 \bullet S_{3,j})$$

$$S'_{2,j} = (3 \bullet S_{0,j}) \oplus S_{1,j} \oplus S_{2,j} \oplus (2 \bullet S_{3,j})$$

The following is an example of Mix Columns:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

The first column of the above example will be verified now. In GF(2^8), addition can be implemented by bitwise XOR operation and multiplication by a value (i.e., by {02}) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with(00011011if the leftmost bit of the original value (prior to the shift) is 1.

Thus, to verify the MixColumns transformation on the first column, the following equations must be verified:

$$(\{02\} \bullet \{87\}) \oplus (\{03\} \bullet \{6E\}) \oplus \{46\} \oplus \{A6\} = \{47\}$$

$$\{87\}) \oplus (\{02\} \bullet \{6E\}) \oplus (\{03\} \bullet \{46\}) \oplus \{A6\} = \{37\}$$

$$\{87\}) \oplus \{6E\} \oplus (\{02\} \bullet \{46\}) \oplus (\{02\} \bullet \{A6\}) = \{94\}$$

$$(\{03\} \bullet \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \bullet \{A6\}) = \{ED\}$$

For the first equation,

$\{02\} \bullet \{87\} = (0000\ 1110) \oplus (0001\ 1011) = (0001\ 0101)$; and

$\{03\} \bullet \{6E\} = \{6E\} \oplus (\{02\} \bullet \{6E\}) = (0110\ 1110) \oplus (1101\ 1100) = (10110010)$.

Then,

$\{02\} \bullet \{87\} = 0001\ 0101$

$\{03\} \bullet \{6E\} = 1011\ 0010$

{46} = 0100 0110

{A6} = 1010 0110

0100 0111

The other equations can be similarly verified.

The inverse mix column transformation, called InvMixColumns, is defined by the following matrix multiplication:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix} \tag{3.5}$$

It is not immediately clear that Equation (3.5) is the inverse of equation (3.3). To show that

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} & S_{0,2} & S_{0,3} \\ S_{1,0} & S_{1,1} & S_{1,2} & S_{1,3} \\ S_{2,0} & S_{2,1} & S_{2,2} & S_{2,3} \\ S_{3,0} & S_{3,1} & S_{3,2} & S_{3,3} \end{bmatrix}$$

$$= \begin{bmatrix} S'_{0,0} & S'_{0,1} & S'_{0,2} & S'_{0,3} \\ S'_{1,0} & S'_{1,1} & S'_{1,2} & S'_{1,3} \\ S'_{2,0} & S'_{2,1} & S'_{2,2} & S'_{2,3} \\ S'_{3,0} & S'_{3,1} & S'_{3,2} & S'_{3,3} \end{bmatrix}$$

This is equivalent to showing that

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

That is, the inverse transformation matrix times the forward transformation matrix equals the identity matrix. To verify the first column of Equation (3.6), the following equations must be verified.

$$(\{0E\} \bullet \{02\}) \oplus (\{0B\} \bullet \{0D\}) \oplus \{09\} \oplus \{03\} = \{01\}$$

$$(\{09\} \bullet \{02\}) \oplus (\{0E\} \bullet \{0B\}) \oplus \{0D\} \oplus \{03\} = \{00\}$$

$$(\{0D\} \bullet \{02\}) \oplus (\{09\} \bullet \{0E\}) \oplus \{0B\} \oplus \{03\} = \{00\}$$

$$(\{0B\} \bullet \{02\}) \oplus (\{0D\} \bullet \{09\}) \oplus \{0E\} \oplus \{03\} = \{00\}$$

For the first equation, $\{0E\} \cdot \{02\} = 0001\ 1100$; and $\{09\} \cdot \{03\} = \{09\} \oplus (\{09\} \cdot \{02\}) = 00001001 \oplus 00010010 = 00011011$.

Then

$$\begin{aligned} \{0E\} \bullet \{02\} &= 0001\ 1100 \\ \{0B\} &= 0000\ 1011 \\ \{0D\} &= 0000\ 1101 \\ \{09\} \bullet \{03\} &= 0001\ 1011 \\ &\quad\ 0000\ 0001 \end{aligned}$$

The other equations can be similarly verified.

The coefficients of the matrix in Equation (3.3) are based on a linear code with maximal distance between code words, which ensures a good mixing among the bytes of each column. The mix column transformation combined with the shift row transformation ensures that after a few rounds, all output bits depend on all input bits.

In addition, the choice of coefficients in MixColumns, which are all {01}, {02}, or {03}, was influenced by implementation considerations. As was discussed, multiplication by these coefficients involves at most a shift and an XOR. The coefficients in InvMixColumns are more formidable to implement. However, encryption was deemed more important than encryption for two reasons:

1) For the CFB and OFB cipher modes, only encryption is used.
2) As with any block cipher, AES can be used to construct a message authentication code, and for this only encryption is used.

3.6 Add Round Key Transformation

In the forward add round key transformation, called AddRoundKey, the 128bits of State are bitwise XORed with the 128 bits of the round key. As shown in Figure3.4 (b), the operation is viewed as a column wise operation between the 4 bytes of a State column and one word of the round key; it can also be viewed as a byte-level operation. The following is an example of AddRoundKey:

47	40	A3	4C	⊕	AC	19	28	57	=	EB	59	8B	1B
37	D4	70	9F		77	FA	D1	5C		40	2E	A1	C3
94	E4	3A	42		66	DC	29	00		F2	38	13	42
ED	A5	A6	BC		F3	21	41	6A		1E	84	E7	D2

The first matrix is State, and the second matrix is the round key.

The inverse add round key transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse.

The add round key transformation is as simple as possible and affects every bit of State. The complexity of the round key expansion, plus the complexity of the other stages of AES ensure security.

3.7 AES Key Expansion

The AES key expansion algorithm [11] takes as input a 4-word (16-byte) key and produces a linear array of 44 words (156 bytes). This is sufficient to provide a 4-wordround key for the initial Add Round Key stage and each of the 10 rounds of the cipher.

The following pseudocode describes the expansion:

$key\ expansion\ (byte\ key[16], word\ w[44])$

{

$word\ temp$

$for\ (i = 0\ ; i < 4\ ; i++)$

$w[i] = (key\ [4 * i], key\ [4 * i + 1], key\ [4 * i + 2], key\ [4 * i + 3]\ );$

$for\ (i = 0\ ; i < 4\ ; i++)$

{

$temp = w\ [i - 1];$

$if(i\ mod\ 4 = 0) temp = Subword\ \left(Rotword(temp)\right)^{Rcon\left[\frac{i}{4}\right]};$

$w[i] = w[i - 4]^{temp}$

}

}

The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time. Each added word w[i] depends on the immediately preceding word, w [i - 1], and the word four positions back, w[i - 4]. In three out of four cases, a simple XOR is used. For a word whose position in the w array is a multiple of 4, a more complex function is used.

Figure 3.7 illustrates the generation of the first eight words of the expanded key, using the symbol g to represent that complex function.

The function g consists of the following sub functions:

1) **RotWord** performs a one-byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].
2) **SubWord** performs a byte substitution on each byte of its input word, usingthe S-box (Table 3.2a).
3) The result of steps 1 and 2 is XORed with a round constant, Rcon[j].

The round constant is a word in which the three rightmost bytes are always ‘0’. Thus the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word.

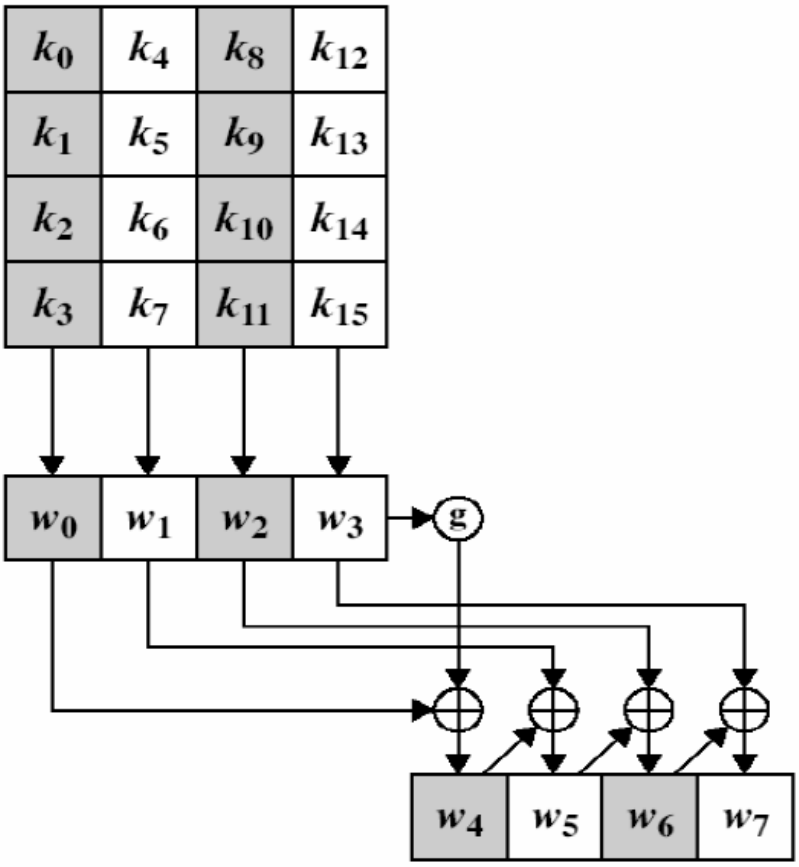


Figure 3.7 AES key expansion [17]

The round constant is different for each round and is defined as Rcon [j] = (RC [j], 0, 0, 0), with RC [1] = 1, RC [j] = 2 • RC [j - 1] and with multiplication defined over the field GF (2^8).

The values of RC [j] in hexadecimal are:

J	1	2	3	4	5	6	7	8	9	10
RC[jJ	01	02	04	08	10	20	40	80	1B	36

For example, suppose that the round key for round 8 is:

EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

Then the first 4 bytes (first column) of the round key for round 9 are calculated as follows:

i decimal	temp	After RotWord	After SubWord
36	7F8D292F	8D292F7F	SDAS1SD2
Rcon (9)	After XORwith Rcon	w[i - 4J	w[iJ= temp ⊕ w[i - 4]
1B000000	46ASlSD2	EAD27321	AC7766F3

The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks. The inclusion of a round-dependent round constant liminates the symmetry, or similarity between the ways in which round keys are generated in different rounds. The specific criteria that were used are [13] as follows:

- Knowledge of a part of the cipher key or round key does not enable calculation of many other round key bits.
- An invertible transformation [i.e., knowledge of any *Nk* consecutive words of the Expanded Key enables regeneration the entire expanded key (*Nk* = Keysize in words).
- Speed on a wide range of processors.
- Usage of round constants to eliminate symmetries.
- Diffusion of cipher key differences into the round keys; that is, each key bit affects many round key bits.
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only.
- Simplicity of description.

3.8 Equivalent Inverse Cipher

The AES decryption cipher is not identical to the encryption cipher (Figure 3.1), That is, the sequence of transformation for decryption differs from that for encryption, although the form of the key scheduled for encryption and decryption is the same.

This has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption. There is, however, an equivalent version of the decryption algorithm that has the same structure as the encryption algorithm. The equivalent version has the same sequence of transformation as the encryption algorithm (with transformations replaced by their inverses). To achieve this equivalence, a change in key schedule is needed.

Two separate changes are needed to bring the decryption structure in line with the encryption structure. An encryption round has the structure SubBytes, ShiftRows, MixColumns, Add RoundKey. The standard AES decryption round has the structure InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns. Thus, the first two stages of the AES decryption round need to be interchanged and the second two changes of the decryption round need to be interchanged.

3.9 Concluding Remarks

Advanced Encryption Standard, or AES, is a block cipher adopted as an encryption standard by the US government, and is expected to be used worldwide and analysed extensively, as was the case with its predecessor, the Data Encryption Standard (DES). AES was developed by two Belgian cryptographers, Joan Daemenand Vincent Rijmen, and was based on their previous design, Square. It is also known by the name of "Rijndael",. AES is not precisely Rijndael, as Rijndael supports a larger range of block and key sizes); AES has a fixed block size of 128 bits and a key size of 128, 192 or 256 bits. Unlike its predecessor DES, it is not a Feistel network, but a substitution-permutation network. AES is fast in both software and hardware, is relatively easy to implement, and requires little memory. As a new encryption standard, it is currently being deployed on a large scale.

As of October 2002, there are no known successful attacks against AES. NSA reviewed all the AES finalists, including Rijndael, and stated that all of them were secure

enough for US Government non-classified data. In June 2003, the US Government announced [14] that AES may be used for classified information:

> "The design and strength of all key lengths of the AES algorithm (i.e., 128,192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 keylengths. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use."

The most common way to attack block ciphers is to try various attacks on versions of the cipher with a reduced number of rounds. AES has 10 rounds for 128-bit keys, 12rounds for 192-bit keys, and 14 rounds for 256-bit keys. As of 2004, the best known attacks are on 7 rounds for 128-bit keys, 8 rounds for 192-bit keys, and 9 rounds for256-bit keys [15].

CHAPTER 4

THE RC6 AS A CANDIDATE TO THE AES

4.1 INRODUCTION

The RC6 Block Cipher algorithm was submitted to the National Institute of Standards (NIST) for consideration as the new Advanced Encryption Standard (AES) in 1998. RC6 was in sharp competition with other algorithms and lost to the Rijndael algorithm, which is now the AES. The RC6 algorithm is presented and some of the issues, as well as advantages, are being addressed.

In this chapter there is first an introduction the RC6 algorithm, with a brief overview and have the major features. This chapter has covered more details how RC6 actually works in terms of key expansion, encryption and decryption. Then a short description of how the algorithm applies to the requirements given by NIST.

4.2 Overview and Major Features

RC6 is a block cipher based on Feistel rounds [18], not Feistel rounds that operates on the two halves of the block, but operates between the pairs of quarters of the block. A block cipher is an encryption algorithm that transforms a block of unencrypted text, also called plaintext, with a fixed length into a block of encrypted text, called ciphertext, with the same fixed length. The RC6 block cipher is very flexible in the way that the number of rounds, size of the key and block, are flexible and can be changed with relatively little effort.

RC6 was designed by Ron Rivest (hence the RC as in Ron's Code) in collaboration with his associates from RSA Laboratories. It is an evolutionary enhancement and improvement over the RC5 block cipher developed by the same company. RC6 makes use of data-dependent rotations and in order to be AES compliant the block cipher must handle 128-bit input and output blocks. Several new features are included in RC6, which were not in the RC5. The cipher uses four 32-bit registers for operations instead of two, as in RC5, which makes it possible to do two rotations per round. Integer multiplication is new as well and increases the diffusion achieved. This again leads to higher security, fewer rounds needed and an increased throughput. A version of RC6 is specified as RC6-w/r/b where w is the word size in bits, r is the number

of rounds we encrypt and finally b is the length of the key we use to encrypt in 8-bit bytes. In the case of RC6 submitted for AES as a candidate the parameter are: w = 32, r = 20 and b = 16, 24 or 32 for respectively 128, 196 and 256 bit keys.

The notations used for RC6 encryption and decryptions are:

1) $a + b$: Addition of a and b modulo 2w.
2) $a - b$: Subtraction of a and b modulo 2w.
3) $a \oplus b$: Exclusive-or of a and b (w-bit word sizes).
4) $a \times b$: Multiplication of a and b modulo 2w.
5) $a \ll b$: Rotate a to the left by the least significant $\log_2$w bits of b.
   For example, the w is in this case 32, and $log_2 32 = 5$, so we shift a using the 5 least significant bits of b.
6) $a \gg b$: We rotate just as described above, however this time to the right.

4.2.1 Key Expansion

From the user supplied key with the length of b a number of sub keys are derived. If the key is not long enough it is padded with zero bytes so the requirements to the length is reached. These sub keys are loaded into an array of c w words L [0,..., c-1], that is the first byte is stored in L[0] and the high order byte, which could be padded with zero bytes, goes into L[c-1].

Now the sub keys are ready to be generated. The keys that will be generated are stored into another array S [0,..., 2r + 3]. The size of this array is 2r + 4, and in the case of the AES candidate that is$2 \times 20 + 4 = 44$.

RC6 uses, just as its predecessor the RC5, two "magic" constants called P_w and Q_w. P_w is derived from the binary expansion of e − 2, where e is the base of the natural logarithm and Qw is derived from the binary expansion of ∅ −1, where ∅ (or Phi) is the Golden Ratio. Other values, however, can be used to make other versions of the algorithm [20,21].

The key expansion procedure can be described as:

$S[0] = P_W$

$for\ i = 1\ to\ 2r + 3\ do$

$S[i] = S[i - 1] + Q_W$

$A = B[i] = j = 0$

$v = 3 \times \max\{c, 2r + 4\}$

$for\ s = 1\ to\ v\ do$

{

$A = S[i] = (S[i] + A + B) \oplus 3$

$B = L[j] = (L[j] + A + B) \oplus (A + B)$

$i = (i + 1) mod\ (2r + 4)$

$j = (J + 1) mod\ c$

}

4.2.2 Encryption

The encryption algorithm in RC6 is relatively simple. The plaintext, which is the input, is stored in four w-bit input registers called (A, B, C, D). Keys are being stored into an array S [0,..., 2r + 3] as described in section 4.1. The ciphertext is the output and is being stored in (A, B, C, D).The encryption algorithm consists of following steps:

RC6 begins with two initial steps:

1) B is added with the subkey S [0]
2) D is added with the subkey S [1]

Every round uses two subkeys; for each round i up to r the subkeys S [2i] and S[2i +1] are being used, that is the first round uses S[2] and S[3].

A round can be described as:

B and D are using the function $f(x) = x(2x + 1) \oplus log_2 w$, which means that

$x\ (2x + 1)$ Is left-shifted 5 bits (or $log_2 w$ where w = 32); A = A ⊕ f (B) which is left-shifted with f (D) and added S [2i]; C = C ⊕ f (D) which is left-shifted with f (B) and added S [2i + 1].

The four quarters in the block are being rotated as:

(A, B, C, D) = (B, C, D, A)

After the last round then:

A is added with subkey S [2r + 2];

C is added with subkey S [2r + 3]

The procedure for encryption in RC6 can be described as:

$B = B + S[0]$

$D = D + S[1]$

$for\ i = 1\ to\ r\ do$

{

$t = (B\ \times (2B + 1)) << log_2 w$

$u = \big(D\ \times (2D + 1)\big) \ll log_2 w$

$A = \big((A\ \oplus t) \ll u\big) + S[2i]$

$C = \big((C\ \oplus u) \ll t\big) + S[2i + 1]$

$(A, B, C, D) = (B, C, D, A)$

}

$A = A + S[2r + 2]$

$C = C + S[2r + 3]$

4.3 Decryption

Decryption works in a similar way as encryption. The difference is that cipher text is the input and plaintext is the output. The use of keys and rounds is the same as for encryption described in section 4.2.The decryption algorithm consists of following steps:

RC6 begins with two initial steps:

1) C is subtracted with the subkey S [2r + 3].
2) A is subtracted with the subkey S [2r + 2].

Every round uses two subkeys; for each round i down to 1 the subkeys S [2i] and S [2i + 1] are being used, that is the first round uses S [21] and S [20].

A round can be described as:

The four quarters in the block are being rotated as:

(A, B, C, D) = (D, A, B, C)

D and B are using the same function as in described in the encryption, Which is $f(x) = x(2x + 1) \ll log_2 w$; C = C – S [2i + 1] which is right-shifted with f (B) and the result is Xor'ed with f (D) A = A – S [2i] which is right-shifted with f (D) and the result is Xor'ed with f (B). After the last round then, the subkey S [1] is subtracted D; the subkey S [0] is subtracted B.

4.4 Applicability to AES Requirements

RC6 is the most elegant end well understood candidate of the candidates submitted to NIST [22], this due to its clean and simple design. We will in the following describe RC6 using some of the evaluation criteria given by NIST.

4.4.1 Security

Security is the most vital criteria that must be fulfilled; it cannot be neglected in any way and it cannot be prioritised lower than any of the other criteria. In the following we will present some of the security aspects in RC6.

4.4.2 Randomness

Although RC6 is efficient and simple it may be vulnerable to statistical attacks [16, 18]. The existence of a statistical weakness in RC6 makes it possible to launch a distinguisher attack as it is possible to distinguish RC6 from a random permutation. This may lead to, as it often does with a distinguisher attack, a key-recovery attack [18].

4.4.3 Soundness

RC6 is based on a long tradition in encryption algorithms and the basics have are well studied (for over 6 years). The predecessors have been used in a variety of applications, most important the RC4 in Secure Sockets Layer (SSL). The issues identified in the earlier versions of these algorithms have therefore been addressed in RC6.

4.4.3 Other security factors

Many processors running in a sequential mode have data-dependent execution times for multiplication and rotation. Research has indicated this to be a vulnerability as it could be leaking key information making a timing attack possible. This may be the case in RC6 [22] just as it was the case in RC5 [19].

However, this may only be a problem for processors running in a sequential mode such as e.g. 8-bit smart-cards. It has, furthermore, been rejected as a potential threat in both RC6 and RC5 by Continue et. al. in [17]. They claim that the encryption and decryption time for RC6 and RC5 can be considered to be data independent, which eliminates the risk of a timing attack. This claim comes from the same people inventing the RC6 so it may be biased in some way.

4.4.4 Cost

It is possible to make a ultra-secure encryption algorithm, but if it is unreasonable expensive compared to what is gained other algorithms should be considered. We will in the following describe RC6 in relation to the costs of using this algorithm.

4.4.5 Licensing requirements

Even though RSA Laboratories holds the patent on RC6 and uses it as a trademark they will not require licensing or royalty as to their policy [17] just as the documentation is released for all to use.

4.4.6 Computational efficiency

Schneier reports in [22] that on the target platform NIST chose (Pentium Pro 200 MHz) a performance on 250 clock cycles per block en- or decrypted was achieved which makes it undoubtedly the fastest algorithm on the target platform. However, on other architectures this is not the general picture, in fact RC6 is almost three times slower on a Pentium than a Pentium Pro [22], although it is still among the better performing candidates [17].

4.4.7 Memory requirements

Over 210 bytes of RAM (176 bytes of subkeys, 16 bytes of plaintext and 16 bytes of key plus additional variables) is required and therefore cannot fit on most smart-card CPU's [22]. Another issue is that the subkeys are not generated on the fly but all at once. This precipitation requires 132 clocks cycles compared to 20-40 clocks for processing a block [22]. The key agility is therefore relatively low.

4.5 Algorithm and Implementation Characteristics

In this section we will take a closer look at the application of the algorithm, whether it is useful and flexible enough and how simple it is designed.

4.5.1 Flexibility

The RC6 algorithm is relatively flexible as it is fully parameterised. The key size, rounds, and block size can all be changed easily. There are, including these possible changes, many speed and size tradeoffs that can be made in order to relatively simple fit the algorithm to the requirements one may have [22].

4.5.2 Hardware and software suitability

The algorithm is slower on many embedded platforms (Intel 386/486) and is therefore less attractive for low end 32 bit CPU's [22]. Furthermore, most 8 bit architectures do not have a multiplication instruction set which results in poor performance. But no matter if such an instruction set is present or not, RC6 is still not among the fastest candidates [22].

4.5.3 Simplicity

The algorithm is simple and easy to understand. This makes the algorithm well suitable to implement into hardware [22] using e.g. Very high speed integrated circuits Hardware Description Language (VHDL) to implement Integrated Circuits (IC) or more commonly in embedded processors using low or high level languages. However, the use of 32 bit rotations in RC6 makes the algorithm less suitable for smaller and larger word sizes [16].

CHAPTER 5

FPGA OVERVIEW

The two devices which are used in simulation are the extensible user interface protocol (XUP) Virtex-II Pro and Xilinx Spartan – 3E (XC3S500) FPGA kit. The extensible user interface protocol (XUP) Virtex-II Pro (XC2VP30) [23] Development System provides an advanced hardware platform that consists of a high performance Platform FPGA surrounded by a comprehensive collection of peripheral components that can be used to create a complex system and to demonstrate the capability of the Virtex-II Pro Platform FPGA. Some specifications of Virtex-II Pro (XC2VP30) are as following:

1) A Virtex-II Pro FPGA with PowerPC 405 cores
2) Up to 2GB of Double Data Rate SDRAM
3) System ACE controller and Type II Compact Flash connector for FPGA configuration and data storage
4) Embedded Platform Cable USB configuration port
5) High speed Select MAP FPGA configuration from Platform Flash In-System
6) Programmable Configuration PROM
7) Support for “Golden” and “User” FPGA configuration bit-streams
8) On board 10/100 Ethernet PHY device
9) Silicon Serial Number for unique board identification
10) RS-232 DB9 serial port
11) Two PS-2 serial ports
12) Four LEDs connected to Virtex-II Pro I/O pins
13) Four switches connected to Virtex-II Pro I/O pins
14) Five push buttons connected to Virtex-II Pro I/O pins
15) Six expansion connectors connected to eighty Virtex-II Pro I/O pins with over voltage protection
16) High Speed expansion connector connected to forty Virtex-II Pro I/O pins that may be used differentially or single ended
17) AC-97 audio CODEC with audio amplifier and speaker/headphone output and line level output
18) Microphone and line level audio input

19) On board XSGA output, up to 1200 x 1600 at 70Hz refresh
20) Three High Speed Serial Communications ports, two Host ports and one Target port
21) Off board expansion MGT link, with user supplied clock
22) 100MHz system clock, 75MHz SATA clock
23) Provision for user supplied clock
24) On board power supplies
25) Power-on reset circuitry

The Xilinx Spartan – 3E (XC3S500) FPGA kit is used for implementation of encryption algorithms [24]. The Xilinx ISE 9.2i tool is used for the design. Some specifications of Spartan – 3E kit are as following:

1) Up to 232 user – I/O pins
2) Over 10,000 logic cells
3) 2 – line, 16 – character LCD screen
4) PS/2 mouse or keyboard port
5) VGA display port
6) Two 9 – pin RS – 232 ports (DTE/DCE)
7) 50 MHz clock oscillator
8) Chip scope Soft Touch debugging port
9) Eight discrete LEDs
10) Four slide switches
11) Four push-button switches
12) Speed Grade -4
13) FG320 package

5.1 Introduction to FPGA

A Field Programmable Gate Array (FPGA) is a reprogrammable logic device used as reprogrammable alternative to ASIC chip devices. FPGAs can be used for specific operational behaviour, or general purpose CPU functionality depending on the complexity of the device. FPGA applications include DSP applications, imaging, speech recognition, cryptography, hardware emulation and for many other application specific uses. Many

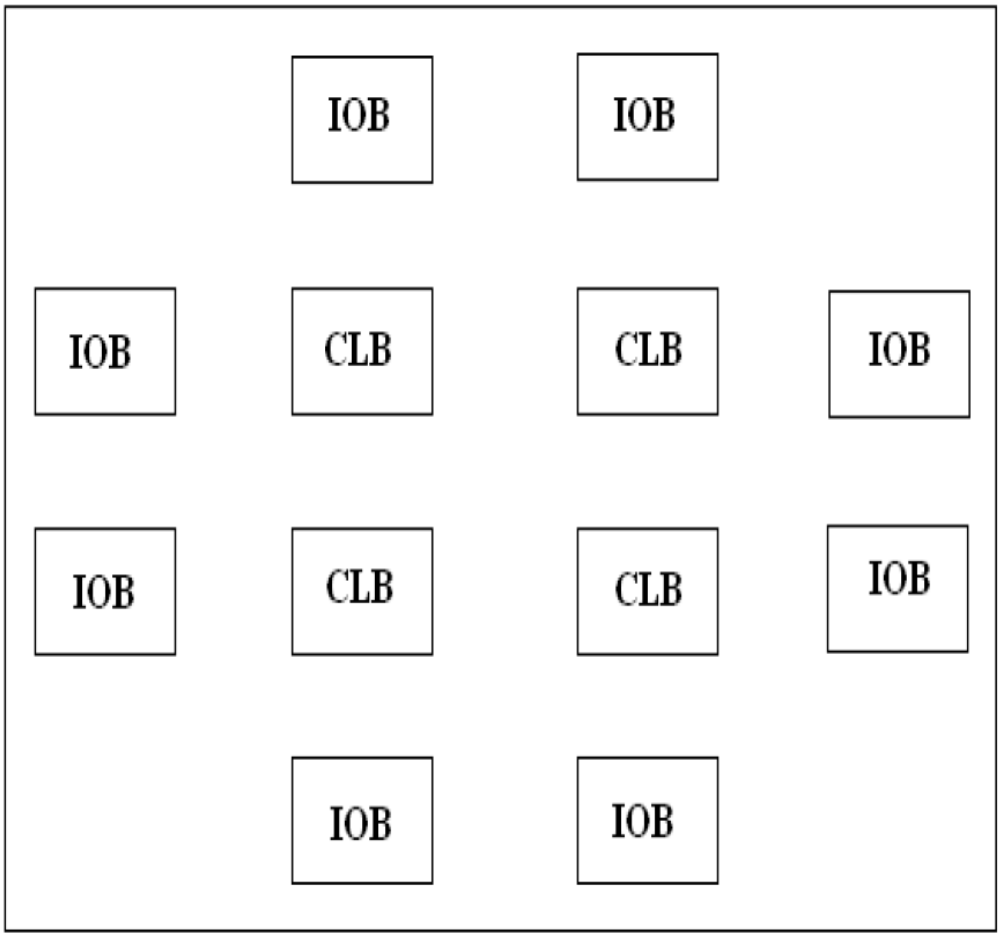


Figure 5.1: FPGA Architecture

FPGAs are implementing shared general purpose CPUs on chip for shorter latency times between operations resulting in higher performance of the overall system.

FPGA is an integrated circuit (IC) that includes a two-dimensional array of:

1) Configurable Logic Blocks (CLBs)
2) Configurable I/O Blocks (IOBs)

FPGAs can be classified as:

1) One time programmable
    - Fuses (destroy internal links with current)
    - Anti-fuses (grow internal links)
    - PROM
2) Reprogrammable
    - EPROM
    - EEPROM

- Flash
- SRAM – volatile

5.2 Design Flow

5.2.1 Design Entity

The HDL languages – Verilog or VHDL are used to design the architecture of the system. A VHDL design can be processed by its entity and architecture declarations. The architecture declaration can be of behavioural, dataflow, or structural modelling style.

5.2.2 Behavioural Simulation

By using behavioural simulation we can verify the functionality of the code designed for our system. Various modifications can be done in case of getting errors in the code.

5.2.3 Synthesis

The synthesis process includes the compilation of the sub-modules in the main module and the analysis of the hierarchy of the design. It checks the syntax of the design gives the output in the net list format which is saved to a Native Generic Circuit (NGC).

5.2.4 Design Implementation

It includes the following three processes:

1. Translation
2. Mapping
3. Place & Route

Translation process compiles all the input net lists and constraints to a logic design file, namely – Native Generic Database (NGD) file having extension .ngd. In this process, the ports are assigned to physical elements such as pins, switches, buttons, etc. This information is stored in the UCF (User Constraint file) file. Mapping process divides the whole circuit into sub-blocks so that they can fit into FPGA blocks. It includes the mapping of input NGD file logic into targeted FPGA elements such as CLBs and IOBs

Place and Route process places the sub-blocks from map process into the logic blocks according to the constraints and connects the logic blocks. This can be done by using Place and Route (PAR) program. This program takes the mapped NCD file as input and gives the completely routed NCD file as output.

5.2.5 Functional Simulation

This simulation process is used to verify the functionality of the design after the design implementation.

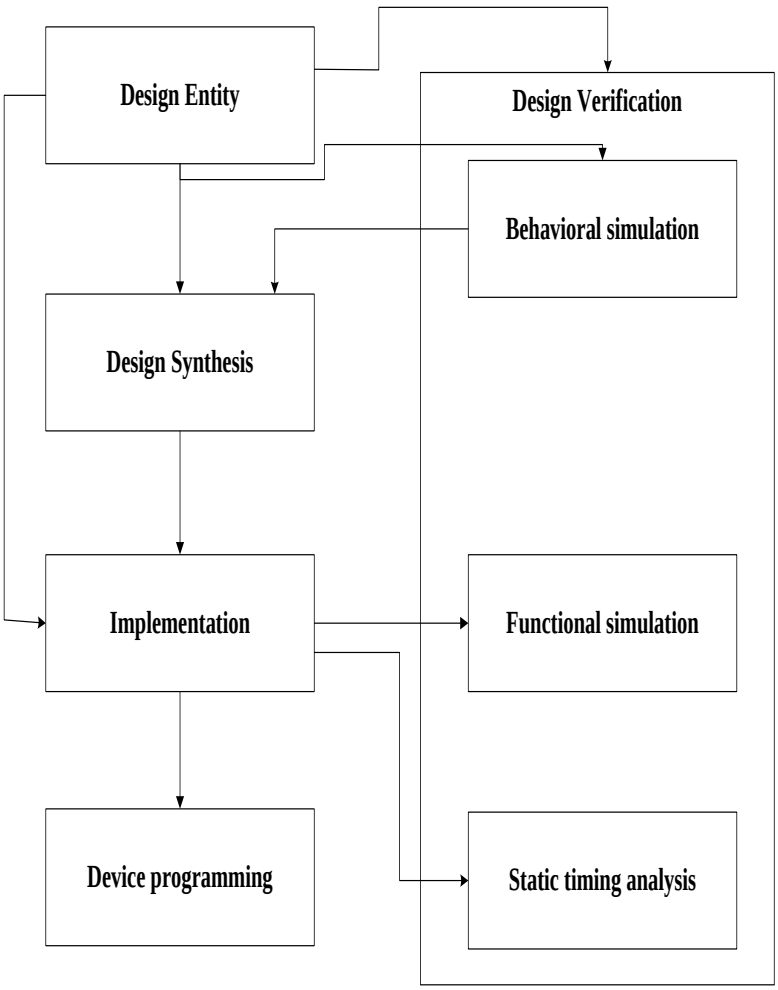


Figure 5.2: FPGA Design Flow

5.2.6 Static Timing Analysis

The static timing analysis can be done by using following three types:

1) Post-fit Static Timing Analysis
2) Post-Map Static Timing Analysis
3) Post-Place and Route Static Timing Analysis

5.2.7 Device Programming

At last the FPGA device is to be programmed by configuring the device. In this process the .bit file is to be assigned to the device which is to be programmed and bye-passing the other devices.

5.3 Overview for implementation of AES

The AES design mentioned in the above chapters is developed and tested adhering to the Xilinx ISE design flow. The tools primarily used are the Xilinx ISE and ModelSim for simulation, synthesis and implementation. For most of testing the Test vectors from the [24] is used. Figure 5.3 shows the design flow implemented. This shows testing at various stages of the design. Initially the Behavioural testing once the Design entry or the Coding part is done, then the Post place and route simulation and after the circuit is implemented the On-chip verification.

For the AES design, a software model of the AES algorithm was initially developed in VHDL which would read a binary text file and then output the encoded bit stream into another binary text file. Once the results from the VHDL code matches with the results in the behavioural simulation the design is further synthesized and checked for behavioural simulation again. The Design is then implemented in a FPGA.

5.3.1 Design Entry

The design entry for this project is basically the VHDL codes for the AES. The top level module for the design is called aes_core.vhd. The design is further divided into 5 parts.

- The AES_fsm_encrypt.vhd
- The key_expansion.vhd
- The sbox.vhd
- The addkey.vhd
- The mix_col.vhd

The algorithmic core is divided into two separate data paths one for encryption and a second for decryption operation. The two data paths are independent, however they share the key_expansion component which provides decrypt and encrypt keys (which are the same only in opposite order).

Each data path is controlled by its own FSM. If configured by the generic DECRYPTION the decryption data path is included and some multiplexers are generated for the shared signals, e.g. result or roundkey_index. For example the encryption data path of aes_core.vhd is given in figure 5.3.

The keyexpansion component computes one column of a roundkey in two clock cycles. In the first cycle the column is substituted through the s-box, in the second cycle the shift-operation is executed. The AES core computes one iteration (round) of the

Rijndael-Algorithm each clock cycle, thus a 128 Bit data block is encrypted or decrypted in 10 cycles plus an initial round.

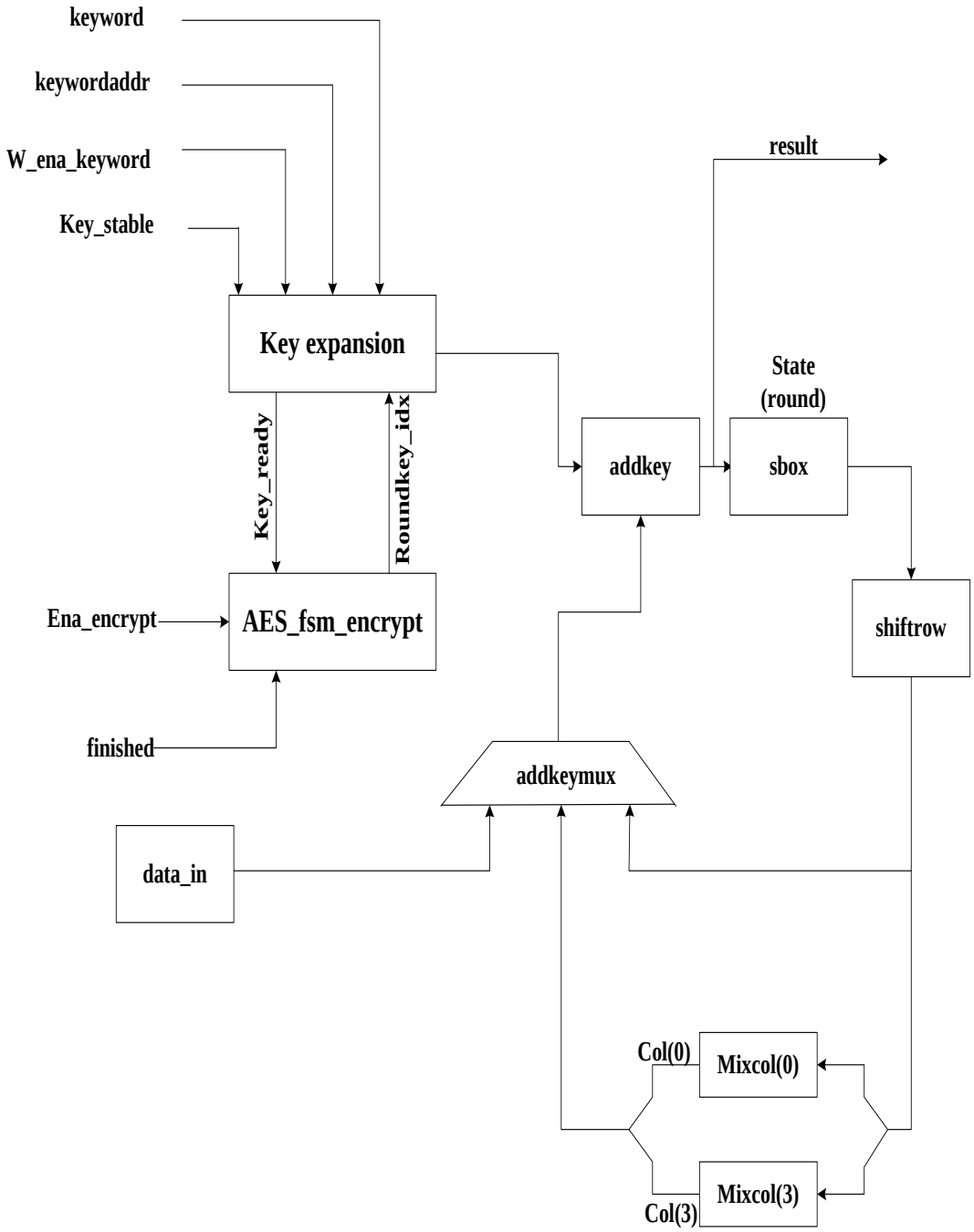


Figure 5.3: Encrypt data path of the AES core as implemented in aes_core.vhd

CHAPTER 6

RESULTS

6.1 Results

6.1.1 Simulation Results for implementation of AES Algorithm

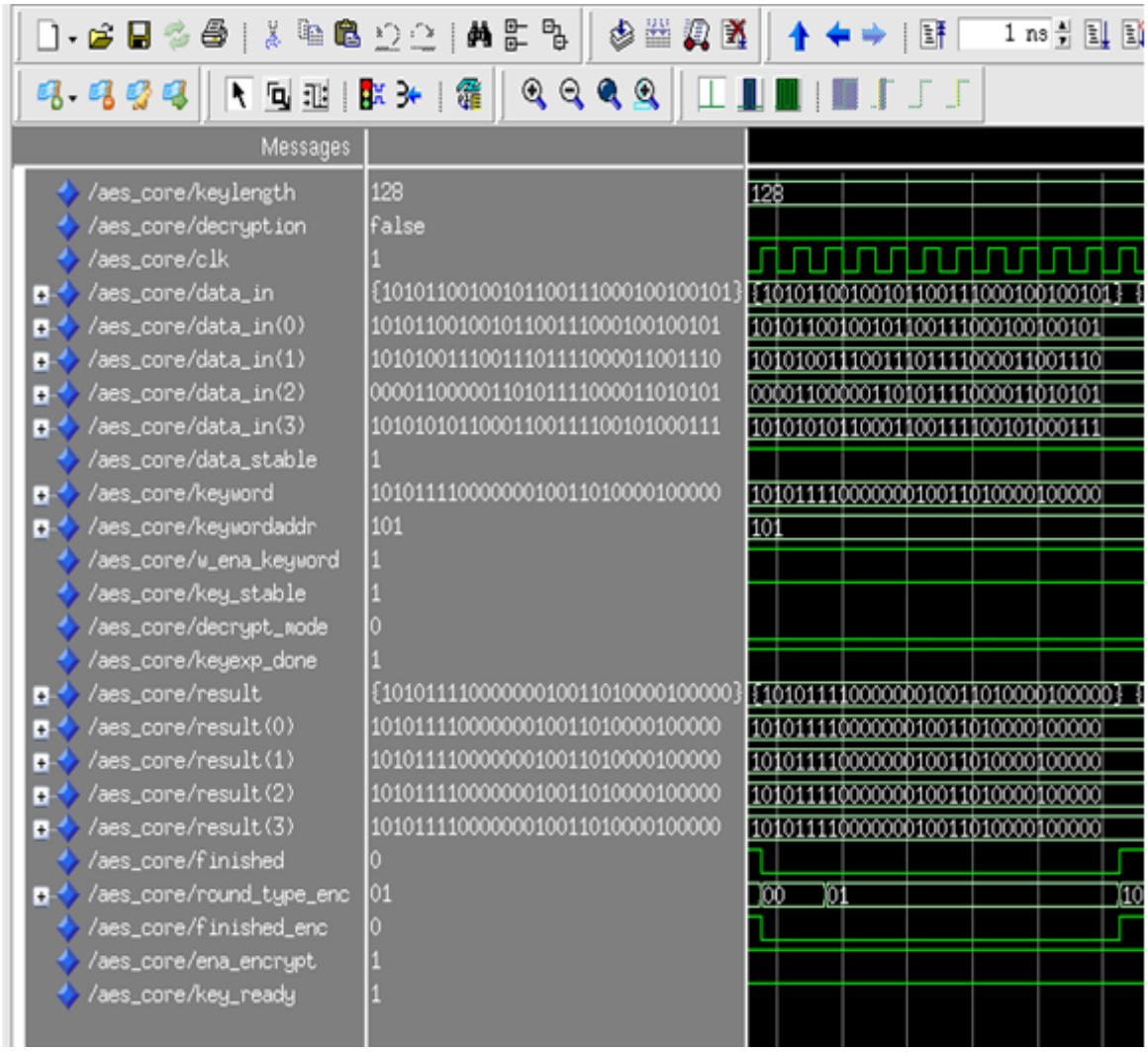


Figure 6.1: Simulation of AES core block

In fig 6.1, 32-bit data is given four times and 4-bit address; so that it gives 128-bit data gets encrypted after processing through the whole encryption process including 10 rounds. The simulation results are shown in figure 6.1, table 6.1 gives the values of various input output signals used in the entity of AES algorithm.

Table 6.1: Values of signals

S.No.	Signal	Value
1.	keylength	128
2.	decryption	False
3.1	data_in(0)	10101100100101100111000100100101
3.2	data_in(1)	10101001110011101111000011001110
3.3	data_in(2)	00001100000110101111000011010101
3.4	data_in(3)	10101010110001100111100101000111
4	data_stable	1
5.	keyword	10101111000000010011010000100000
6.	keywordaddr	101
7.	w_ena_keyword	1
8.	key_stable	1
9.	decrypt_mode	0
10.	keyexp_done	1
11.1	result(0)	10101111000000010011010000100000
11.2	result(1)	10101111000010010011010000100000
11.3	result(2)	10101111100000010011010000100000
11.4	result(3)	10101111000000010011010000100000
12.	finished	1
13.	round_type_enc	10
14.	finished_enc	1
15.	ena_encrypt	1
16.	key_ready	1
17.	ready	1

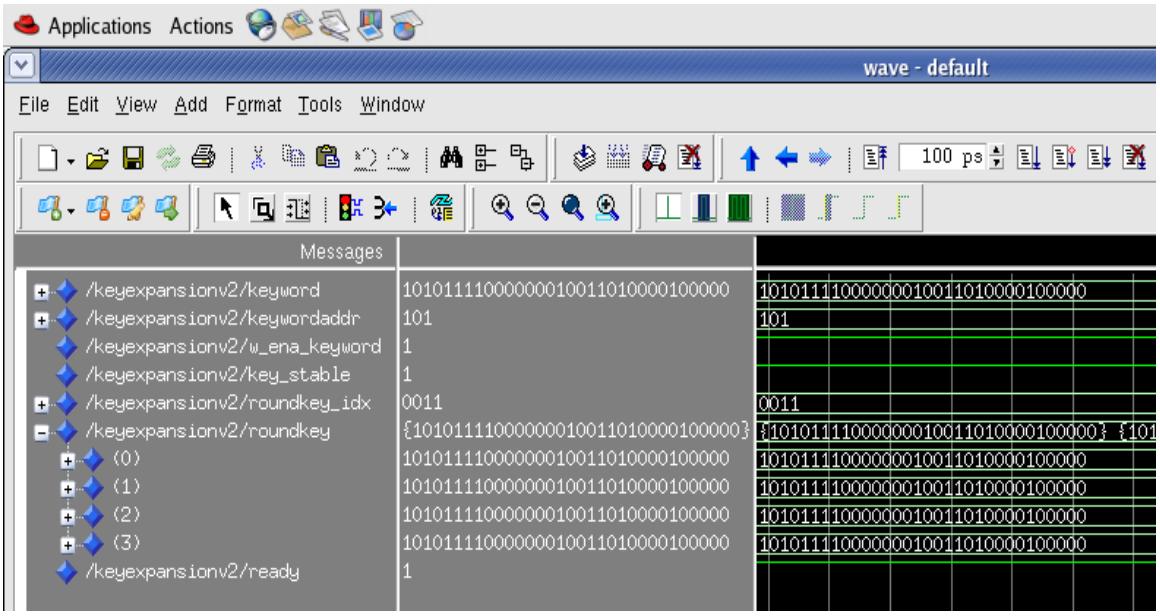


Figure 6.2: Simulation result of key expansion block

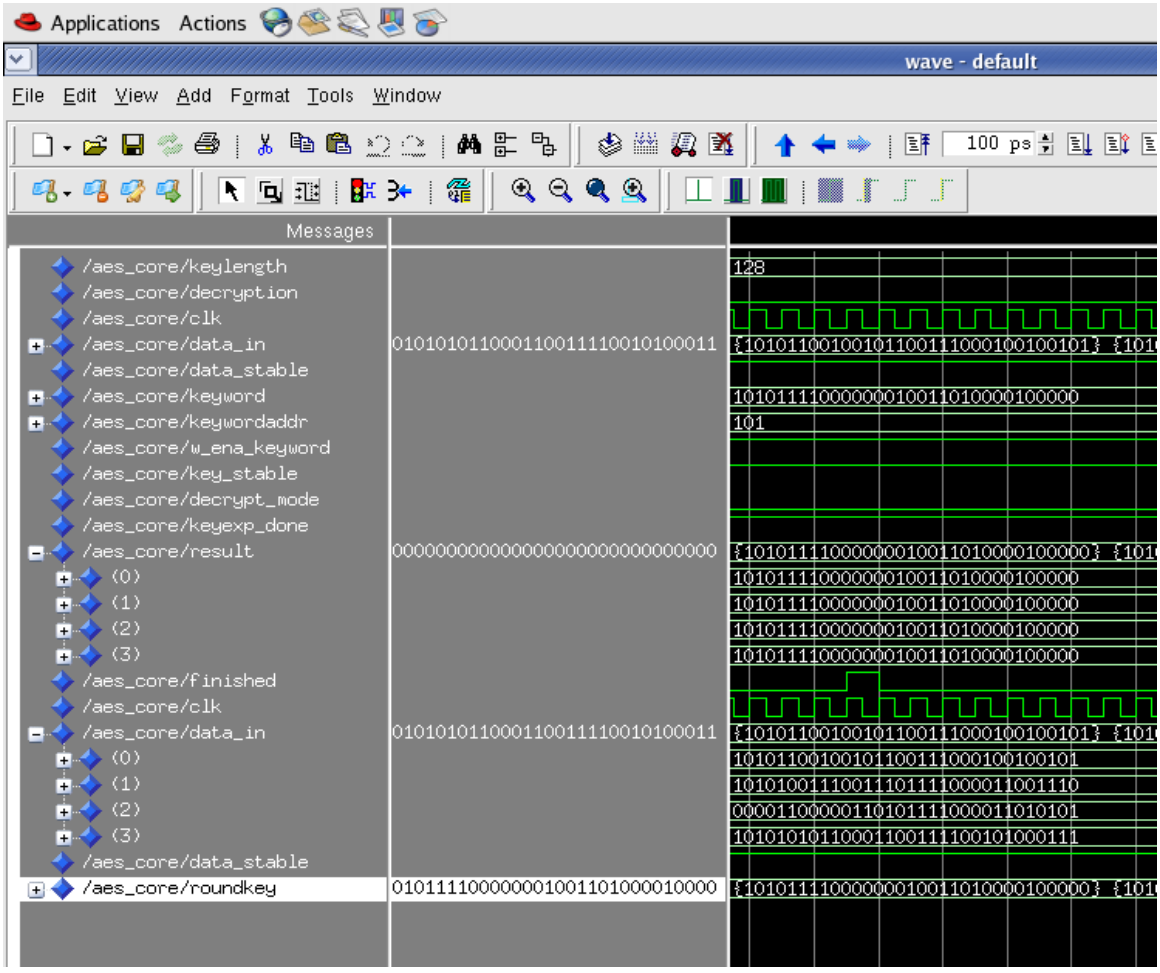


Figure 6.3: Final simulation results after addition of round key

Table 6.2: Description of the signals in AES core block

S.No	Signal	Type
1.	keylength	generic input
2.	decryption	Boolean (true or false) input
3.	data_in	128- bit plaintext input of [4, 32] two dimensional array
4.	data_stable	input for checking complete reception of 128 bits data
5.	keyword	input of key required for encryption
6.	keywordaddr	input for key word address
7.	w_ena_keyword	input for enabling the reception of key
8.	key_stable	input as a check for complete reception of key
9.	decrypt_mode	input to show whether encryption / decryption
10.	keyexp_done	output to show expansion of key into 44 bits
11.	result	output for 128 - bit ciphertext
12.	finished	output for completion of encryption
13.	round_type_enc	internal signal
14.	finished_enc	internal signal set high at completion of encryption
15.	ena_encrypt	internal signal which gets value high on encryption
16.	key_ready	internal signal which gets value high on receiving key

The algorithmic core is divided into two separate data paths one for encryption and a second for decryption operation. The two data paths are independent, however they share the “keyexpansion” component which provides decrypt and encrypt keys (which are the same only in opposite order). Before any AES operation can be started the initial user key has to be written to KEY segment of the memory map. After the user key is transferred to the component the KEY_VALID bit must be set to start the key expansion. This bit can be set simultaneously with DEC or ENC bit of the control register.

Figure 6.4: AES core block

Table 6.3: Synthesis Results of AES core program

S.No.	Parameter	Value
1.	FPGA Device Package	xc2vp30-5-ff896
2.	Number of slices	1,127 out of 13,696
3.	Number of slice Flip Flops	459 out of 27,392
4.	Number of 4 input LUTs	2,029 out of 27,392
5.	Number of IOB flip flops	33
6.	Number of bonded IOBs	75 out of 416
7.	Number of GCLKs	1 out of 16
8.	Minimum period	4.043 ns
9.	Maximum frequency	247.365 MHz
10.	Power(X-power)	399mW
11.	Slack(Setup)	4.866 ns
12.	Slack(Hold)	6.161 ns

6.1.2 Simulation Results for implementation of RC6 Algorithm

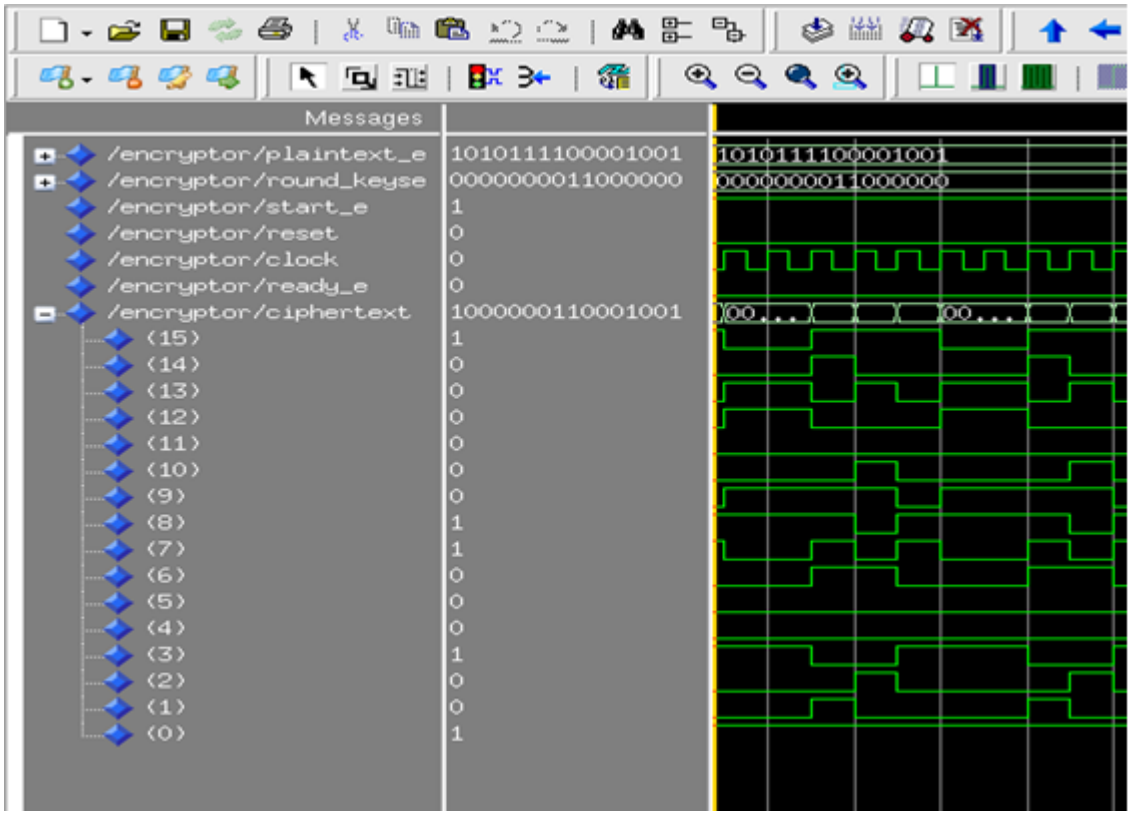


Figure 6.5: Simulation of RC6 core block

Table 6.4: Values of the signals of RC6

S.No	Signal	Value
1.	plaintext_e	1010111100001001
2.	roundkey_se	0000000011000000
3.	start_e	1
4.	reset	0
5.	clock	Rising edge
6.	ready_e	0
7.	ciphertext	10000001100001001

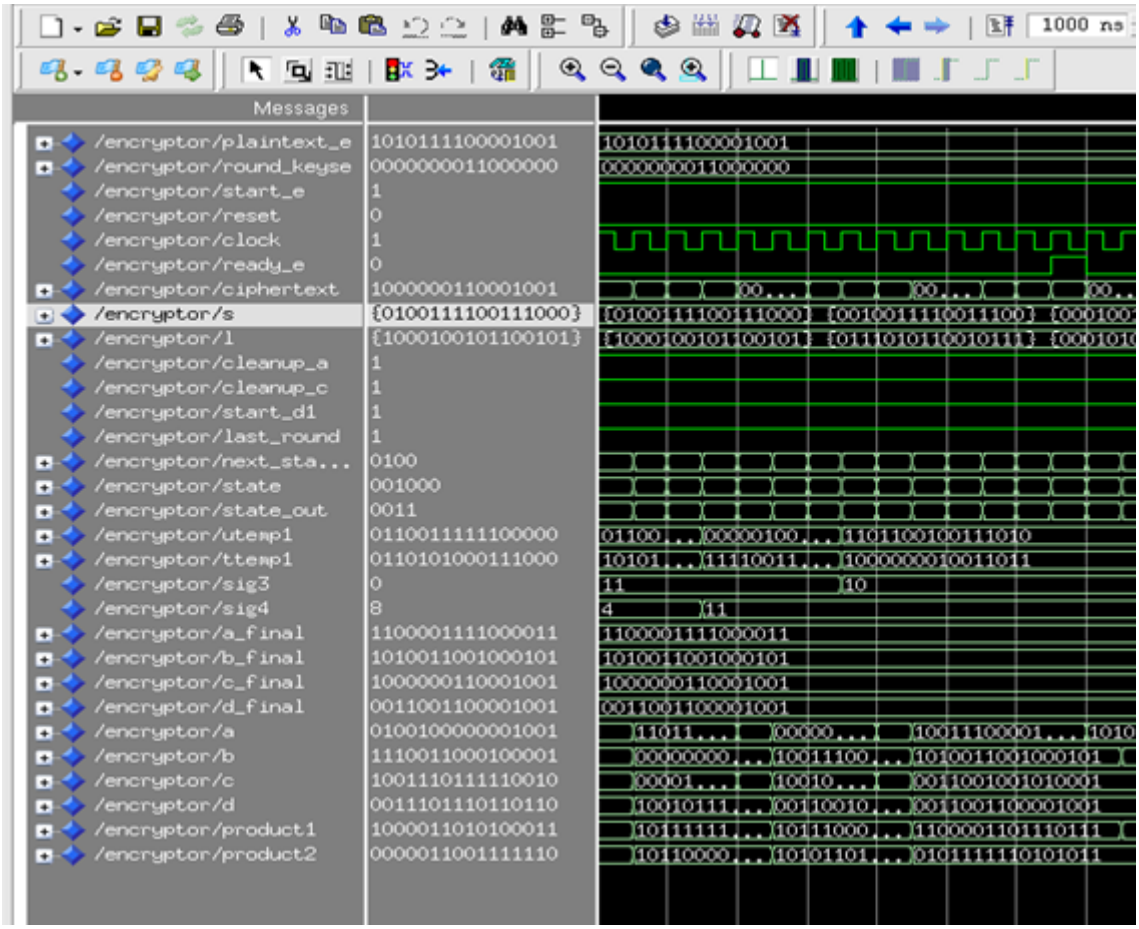


Figure 6.6: Simulation of RC6 core block with all internal signals

Table 6.5: Description of the signals

S.No	Signal	Type
1.	plaintext_e	16-bit input signal to be encrypted
2.	roundkey_se	16-bit input signal
3.	start_e	Input signal to start encryption
4.	reset	Input signal to reset the value
5.	clock	Clock as input
6.	ready_e	Output signal after 93 clock pulses is set high
7.	ciphertext	16-bit output signal as ciphertext

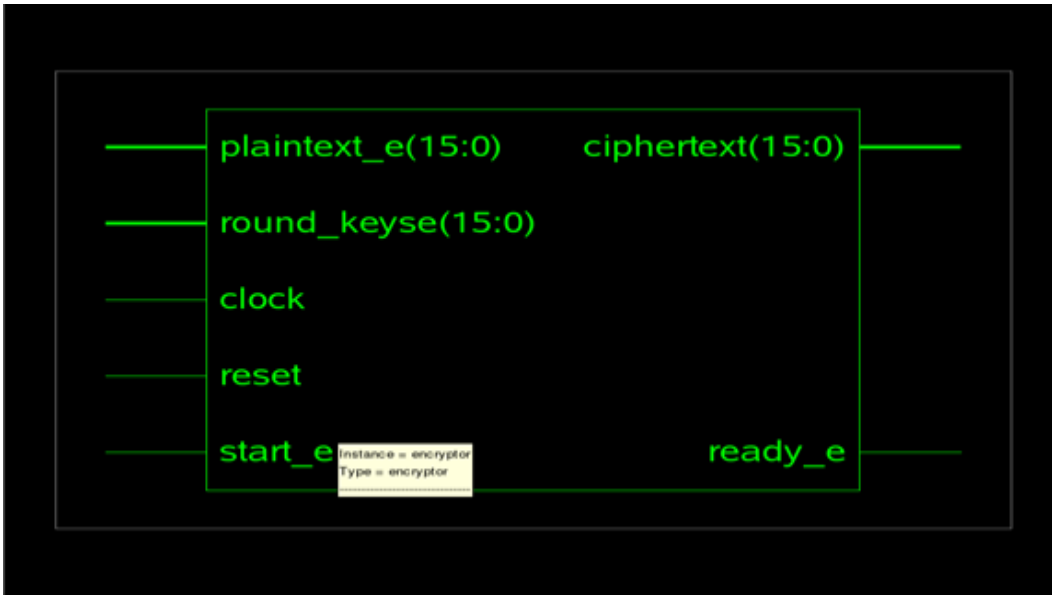


Figure 6.7: RC6 core block

Table 6.6: Synthesis Results of RC6 core program

S.No	Parameter	Value
1.	FPGA Device Package	xc2vp30-5-ff896
2.	Number of slices	1,368 out of 13,696
3.	Number of slice Flip Flops	219 out of 27,392
4.	Number of 4 input LUTs	2,601 out of 27,392
5.	Number of IOB flip flops	16
6.	Number of bonded IOBs	52 out of 416
7.	Number of GCLKs	1out of 16
8.	Minimum period	6.318 ns
9.	Maximum frequency	158.274 MHz
10.	Power(X-power)	325 mW
11.	Slack(Setup)	110.009 ns
12.	Slack(Hold)	3.460 ns

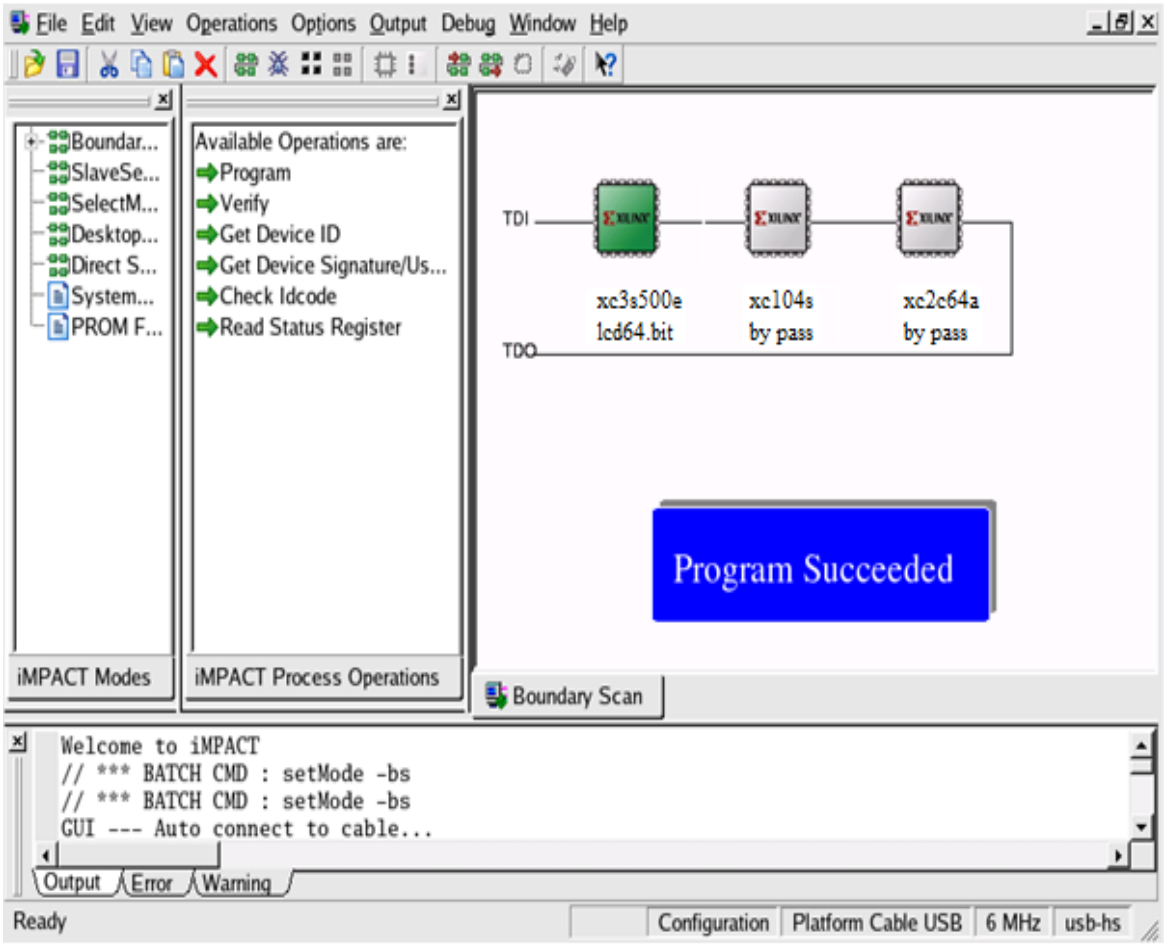


Figure6.8: LCD display (output of 16 – bit RC6 core) on Spartan3E FPGA kit

As shown in Figure 6.8, when programming is complete, the program succeeded message is displayed. Table 6.6 shows synthesis results of LCD interfacing. Figure 6.8 shows output of 16-bit RC6 core on LCD of Spartan kit.

Table 6.7 Comparison of three different architectures of AES algorithm for various performance measures

S.No.	Parameter	AES ARCHITECTURE 16 sbox & 2 dual port RAM	AES ARCHITECTURE 16 sbox & 8 dual port RAM	AES ARCHITECTURE 4 sbox & 2 dual port RAM
1.	**Slice LUTs**	459	801	335
2.	**Delay**	4.043 ns	5.463 ns	6.922 ns
3.	**Power**	399mW	2,107mW	124mW

CHAPTER 7

CONCLUSION

In this thesis, new hardware architectures for the Advanced Encryption Standard (AES) algorithm were presented. FPGA Xilinx technology was used to synthesis the designs and provide post placement results using Xilinx ISE 9.2. After implementing the AES algorithm on a single processor, it is found that the AES algorithm has a linear complexity that means when the value of N (number of data blocks) ranges from 10-100000, the execution time will vary from 10^{-5} to 1 sec (each operation is assumed to take 10^{-6} seconds). However, when the value of N is greater than 10^{8}, the execution time of the algorithm will require several days to encrypt or decrypt. This design took advantage from the repeated operations in each stage of the encryptor to achieve resources merging and sharing. In this encryptor the mix columns step is relocated and all the round keys are obtained in the isomorphic mapping. Here internal pipelining for the composite field S-BOX was applied. This pipelining allowed parallel processing for the state array columns in addition to S-BOX sharing between the main round unit and the key expansion unit. Moreover, this design used for all round keys which prevents using large area to store all the keys in addition to cancelling the extra delay resulting in pre-calculation and storage for all round keys. This architecture has achieved higher FPGA (Throughput/Area) efficiency compared to previous AES designs.

Finally, the RC6 meets the requirements given by NIST to the new AES. RC6 was one of the best candidates for the AES. It proves to be very simple, fast and has a low security margin. However, RC6 was not the best algorithm to suit the requirements given by NIST. It should be mentioned here that this does not imply that RC6 is not a brilliant algorithm, but that it is less suited for some of the more specific requirements. In some areas the RC6 was, and still is, superior to the competitors, so as a final remark it must be mentioned that there still is a place for RC6 as a leading encryption algorithm.

7.1 Future Scope

The research works achieved in this thesis are behind our motivation to present the following recommendations for future research investigations in the hardware design for the AES algorithm and other possible cryptography algorithms:

1. Very high speed universal AES Processor that works on all key sizes could be implemented by getting benefit from the I-BOX technique using a loop unrolled system.
2. Future AES designs with 8-bits data path could be designed based on the S-BOX sharing.
3. Other cryptography algorithms might benefit from the ideas of merging and relocating techniques, especially in loop unrolled systems

REFERENCES

[1] A. Menezes and S.Vanstone "Handbook of Applied Cryptography", CRC Press, 1996.

[2] National Bureau of Standards, NBS FIPS PUB 46, "Data Encryption Standard", U.S. Department of Commerce, January 1977.

[3] D. Coppersmith "The Data Encryption Standard (DES) and Its Strength against Attacks", IBM Journal of Research and Development, No.5, 1994.

[4] W. Diffie and M. Hellman "Multiuser Cryptographic Techniques", AFIPS National Computer Conference, pp.109-112, 1976.

[5] T. Korner "The Pleasures of Counting", Cambridge University Press, England 1996.

[6] D. Khan "The Code breakers: The Story of Secret Writing", New York, 1996.

[7] B. Schneier "Applied Cryptography", New York: Wiley, 1996.

[8] W. Stallings "Cryptography and Network Security: Principles and Practices."Third Edition, Pearson Education, Inc., 2003.

[9] R. Rivest, A. Shamir and L. Adleman "A Method for Obtaining Digital Signature and Public-Key Cryptosystems", Communication of the ACM, 21, No. 2, pp.120-126, 1978.

[10] RSA Laboratories, "The RSA Laboratories Secret-key Challenge: Cryptographic.

[11] J. Daemon and V. Rijmen "The Rijndael Block Cipher: AES Proposal", NIST, Version 2, March 1999.

[12] J. Daemon and V. Rijmen "Rijndael: The Advanced Encryption Standard.", Dr. Dobb's Journal, No. 3, pp. 137-139, 2001.

[13] J. Daemon and V. Rijmen "The Design of Rijndael: The Wide Trail Strategy Explained." New York, Springer, 2000.

[14] National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information, CNSS Policy No. 1, Fact Sheet No. 1, June 2003.

[15] O. Baudron, H. Gilbert, L. Granboulan, H. Handschuh, A. Joux, P. Nguyen, F. Noilhan,D. Pointcheval, T. Pornin, G. Poupard, J. Stern, and S. Vaudenay, "Report on the AES Candidates", in Proceedings of Second AES Candidate Conference (AES-II) Rome, Italy, March 1999.

[16] S. Contini, R. Rivest, M. Robshaw, and Y. Yin, Comments on the First Round AES Evaluation of RC6.

[17] H. Gilbert, H. Handschuh, A. Joux, and S. Vaudenay, "A Statistical Attack on RC6", pp. 64–74, Springer 2001.

[18] W. N. H. Feistel and J. L. Smith, "Some Cryptographic Techniques for Machine-to-Machine Data Communications", IEEE, pp. 1545–1554, 1975.

[19] H. Handschuh and H. M. Heys, "A Timing Attack on RC5", in Selected Areas in Cryptography, pp. 306–318, 1998.

[20] R. L. Rivest, "the RC5 Encryption Algorithm, in Practical Cryptography for Data Internetworks" IEEE Computer Society Press, 1996.

[21] R. L. Rivest, M. J. B. Robshaw, and Y. L. Yin, "RC6 as the AES", in AES Candidate Conference, pp. 337–342, 2000.

[22] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, "Performance Comparison of the AES Submissions", The second AES conference, pp.1–20, 1999.

[23] Xilinx "Virtex-II Pro and Virtex-II Pro XFPGA User Guide", November 2007.

[24] Xilinx "Spartan 3E User Guide", edition 2003.

[25] X. Zhang and K. K. Parhi, "On the Optimum Constructions of Composite Field for the AES Algorithm", TCAS-II, Vol. 53, No.10, pp. 1153-1157, 2006.

[26] J. M. Granado - Criado , M.A Vega - Rodrıguez, J.M. Sanchez-Perez and Juan A. Gomez-Pulido, "A new methodology to implement the AES algorithm using partial and dynamic reconfiguration", Integration, the VLSI Journal 43 ,pp. 72-80, 2010.