Chapter 1

Introduction

Message authentication involves two aspects: Source authentication, which verifies the identity of the source, prevents the acceptance of messages from a fraudulent source and data integrity, which protects the data from modification [23]. One way of achieving authenticity as well as integrity is creation of message authentication code abbreviation as MAC or cryptographic checksum. MAC is a small fixed-size block of data which is computed by using a secret key that share between communicating parties. The Cipher Block Chaining Message Authentication Code (CBC MAC) [4] is a well-known method to generate a message authentication code (MAC) based on a block cipher. However, it is well known that the CBC MAC is not secure unless the message length is fixed. Therefore, several variants of CBC MAC have been proposed for variable length messages. First Encrypted MAC (EMAC) was proposed [12, 13]. It is obtained by encrypting the CBC MAC value by encryption function again with a new key K2. That is:

$EMAC_{k1,k2} (M) = E_{k2}(CBC_{k1} (M))$

Where M is a message, K1 is the key of the CBC MAC and CBC_{k1} (M) is the CBC MAC value of E. Petrank and Rackoff [18] then proved that EMAC is secure if the message length is a positive multiple of n. EMAC requires two key scheduling of the underlying block cipher E. where E represents encryption function. XCBC takes three keys: one block cipher key K1, and two n-bit keys K2 and K3. TMAC is derived from XCBC by replacing $(K2,K3)$ with $(K2 \cdot u,K2)$, where u is some non-zero constant and "·" denotes multiplication in $GF(2^n)$,Where as OMAC is obtained from XCBC by replacing (K_2,K_3) with $(L.u,L.u^2)$ for some non-zero constant u in $GF(2^n)$,where L is given by $E_K(0^n)$[12].

1.1 Motivation

As one of the most popular modes of operations in use, CBC mode encryption has already received extensive security analysis, and its security properties are well known. One might naturally ask why CBC mode encryption is still receiving treatment in the form of this thesis.

The answer is that while traditional analyses performed on CBC mode encryption are mathematically sound and are fundamental and indispensable pieces of work, there exists a gap between their analyses and the real-world performance for data integrity. More recently, researchers have developed very sophisticated models that attempt to encapsulate all physically observable aspects of cryptographic implementations, yet their work tends to be theoretical in outlook. In this thesis, I have taken a step in bridging that gap between theoretical and practical implementation to calculate cycle /byte for comparing of performance. Selecting fast and secure MAC generation algorithm is main issue along with choosing symmetric encryption algorithm. This thesis is concerned with the empirical performance of encryption using block cipher in cipher chaining mode to create message authentication code. This is the main point of motivation for carrying out this work.

1.2 Objective

The objective of this study is to implement and analyze the performance of CBC MAC and its variants using different symmetric cryptographic algorithm like Triple DES and AES in order to gain the efficiency over existing method like Encrypted MAC (EMAC), XCBC MAC, Two-Key CBC MAC (TMAC) [12] and One-Key CBC MAC (OMAC1) [9].In this study Cycle/Byte calculation will be performed.

1.3 Thesis Organization

The rest of the content in this study is organized into subsequent five chapters.

Chapter 2 provides background study required for dissertation. In this chapter the problem of different message authentication code generation algorithms are mentioned, problem statement is formulated and how this study response those issues is mentioned. Chapter 3 contains previous literature related to this work in detail under literature review. Chapter 4 provides an implementation overview of different MAC generation algorithms based on cipher block chaining operation with the two symmetric encryption algorithm, AES and Triple –DES. The implementation details with sample code are provided in this chapter. Chapter 5 includes the analysis of time required for creating message authentication code (MAC) and finally with the help of average time needed for MAC for all candidates algorithm cycle per byte is calculated. The result of the study is shown in tabular form as well as in graphs. Finally, the concluding remarks and further recommendations are outlined in chapter 6.

Chapter 2

Background Study

Today the internet has virtually become the way of doing business as it offers a powerful ubiquitous medium of commerce and enables greater connectivity of disparate groups throughout the world. So it may have many risks like loss of privacy, loss of data integrity, denial of service and identify spoofing. To the solution of these threads in internet many secure cryptographic algorithms are needed for providing services such as confidentiality, data integrity and authentication to handle packets which may vary in size over a large range. The size of the message has a significant impact on the performance of such algorithms. In particular, the message authentication algorithms process messages partitioned into blocks. Hence the messages have to be prepared by padding the required amount of zero bits to get an integer number of blocks. This process becomes a considerable overhead when the short messages are more dominant in the message stream [18]. In this thesis, for simplicity communicating parties are named as Allice and Bob where as attacker named as Darth.

2.1 Problem Definition

Some researchers proposed parallelizable MAC algorithm Bellare, Guerin and Rogaway proposed XOR MAC [7]. Gligor and Donescu proposed XECB –MAC. Black and Rogaway proposed PMAC [19]. However these algorithms have overhead as, XOR MAC requires much more invocations of encryption than the other MAC algorithms. XECB –MAC requires modulo 2^n arithmetic and three more invocations of E than XCBC and TMAC. PMAC needs to generate a sequence of masks. Therefore TMAC and XCBC are better than these algorithms in non-parallelizable environment [10]. Most of the widely used cryptographic algorithm and techniques to make better security are under attack today. Some algorithm are discarded because of other overhead not only by security reason so performance and capability to run in every machine is also should be analyzed. Various CBC MAC Schemes have been in wide use for many years for protecting and guaranteeing the origin of data. That all schemes have been specifically designed for use with message of variable length, with the goal of minimizing the number of block cipher operations required to compute MAC. Key generation is major issues in all variants of CBC MAC for securing variable length of message. Time constant is the important factor for comparing the performance of its variants to create MAC. All of the variants are based on symmetric key encryption scheme. Different symmetric encryption algorithms are used for creating cipher in every step while creating MAC, so performance analysis is needed for selecting best approach of symmetric key encryption algorithm with best variant of CBC MAC.

2.2 Background Study

Since all the study require the basic terms and terminology related to that study. In this context basic study related to this work are outlined in the following sections;

2.2.1 Cryptography

Cryptography is art of protecting information by transforming it *(*encrypting it) into an unreadable format, called cipher text. Only those who possess a secret key can decipher (or decrypt) the message into plaintext. Encrypted messages can sometimes be broken by cryptanalysis, also called code breaking, although modern cryptography techniques are virtually unbreakable. Cryptography enables one to store sensitive information or transmit it across insecure networks so that it cannot be read by anyone except the intended recipient. While cryptography is the science of securing data, cryptanalysis is the science of analyzing and breaking secure communication. Classical cryptanalysis involves an interesting combination of analytical reasoning, application of mathematical tools, pattern finding, patience, determination, and luck. Cryptanalysts are also called attackers and represented as Darth. Cryptology embraces both cryptography and cryptanalysis. The modern cryptography can be divided into two main branches [8, 23]:

- Symmetric Cryptography, where the same key is used to encrypt a message and decrypt data.
- Asymmetric cryptography, where two different keys are used for encryption and decryption.

2.2.1.1 Symmetric Cryptography

Symmetric cryptography is a form of cryptosystem in which encryption and decryption are performed using the same key. It is also known as private key cryptosystem. Symmetric cryptosystem was the only type of encryption technique in use prior to the development of public key cryptosystem. Which can be defined as: Let M denotes the set of all possible plaintext messages, C the set of all possible ciphertext, K the set of all possible keys, k: $M \to C$ is the encryption function, and k: $C \to M$, is decryption function, such that $k(k(m)) = m$ for all $m \varepsilon M$ and $k \varepsilon K$. In this cryptosystem, sender and receiver have to initially agree upon a secret key $k \varepsilon K$. After that, whenever sender wishes to send a message $m \varepsilon M$ to receiver, sender sends the ciphertext $C = k(m)$ to receiver, from which receiver can recover m by applying the decryption function as $m = k(C)$ [8]. The notion of private key cryptosystem is depicted in Figure 2.1.

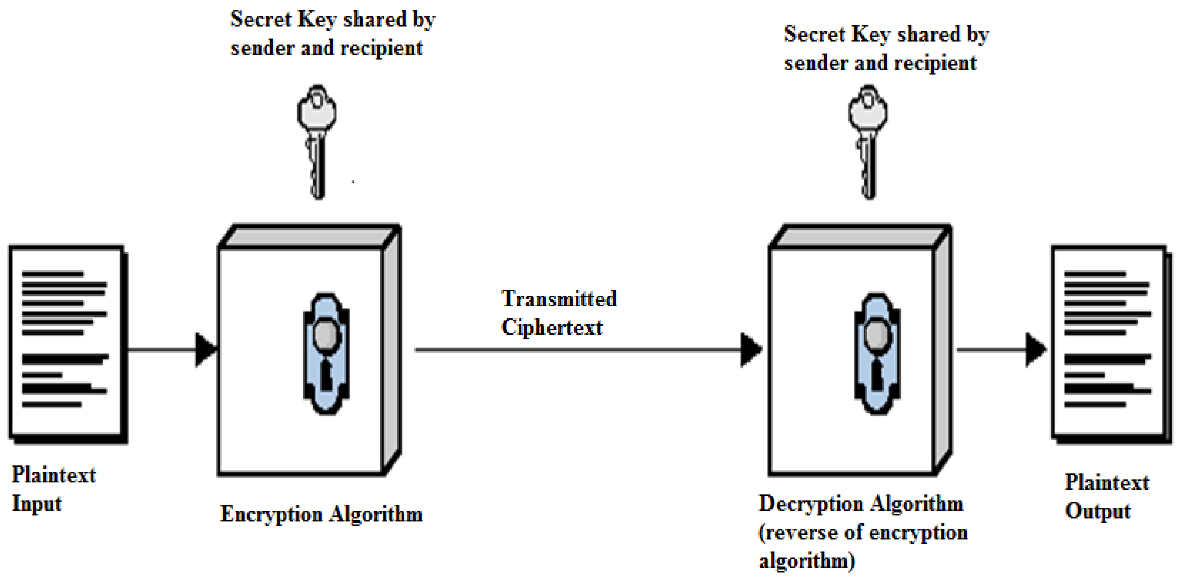

Secret Key shared by sender and recipient
Secret Key shared by sender and recipient
Transmitted Ciphertext
Plaintext Input
Encryption Algorithm
Decryption Algorithm (reverse of encryption algorithm)
Plaintext Output

Figure 2.1: Simplified Model of Symmetric Encryption [23]

The effectiveness of private key cryptosystems relies on the requirement of strong encryption algorithm which would be like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or find out the key and another requirement is that sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. Modern techniques used in private key cryptosystem are XOR Cipher, Rotation Cipher, Substitution Cipher: S-box, Transposition Cipher: P-box, Modern Round Ciphers, Data Encryption Standard (DES), Advanced Encryption Standard (AES) and so on. As mentioned in [23], private key cryptosystems have numerous limitations which are outlined below:

- **Key distribution problem:** Two parties that want to communicate each other need to set up a shared secrete key before starting communicate over an insecure channel.

- **Key management problem:** Every pair of users must share a secret key leading to a total of n*(n-1)/2 keys. Where n be the users in a network. If n is large, then the number of keys become unmanageable and traffic in network may be increased. It makes difficult to manage key.

- **No signatures possible:** A digital signature is an authentication mechanism that enables the creator of the message to attach a special token that acts as a signature. A digital signature allows the receiver of a message to convince any third-party that the message in fact originated from the sender.

2.2.1.2 Asymmetric Cryptography

Asymmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the different key- one a public key and one a private key. It is also known as public-key encryption. Asymmetric encryption transforms plaintext into ciphertext using a one of two keys and an encryption algorithm. Using the paired key and a decryption algorithm, the plaintext is recovered from the cipher text. Asymmetric encryption can be used for confidentiality, authentication, or both. The most widely used public-key cryptosystem is RSA. Public-key algorithms are based on mathematical functions rather than on substitution and permutation. More important, public-key cryptography is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key. The use of two keys has profound consequences in the areas of confidentiality, key distribution, and authentication.

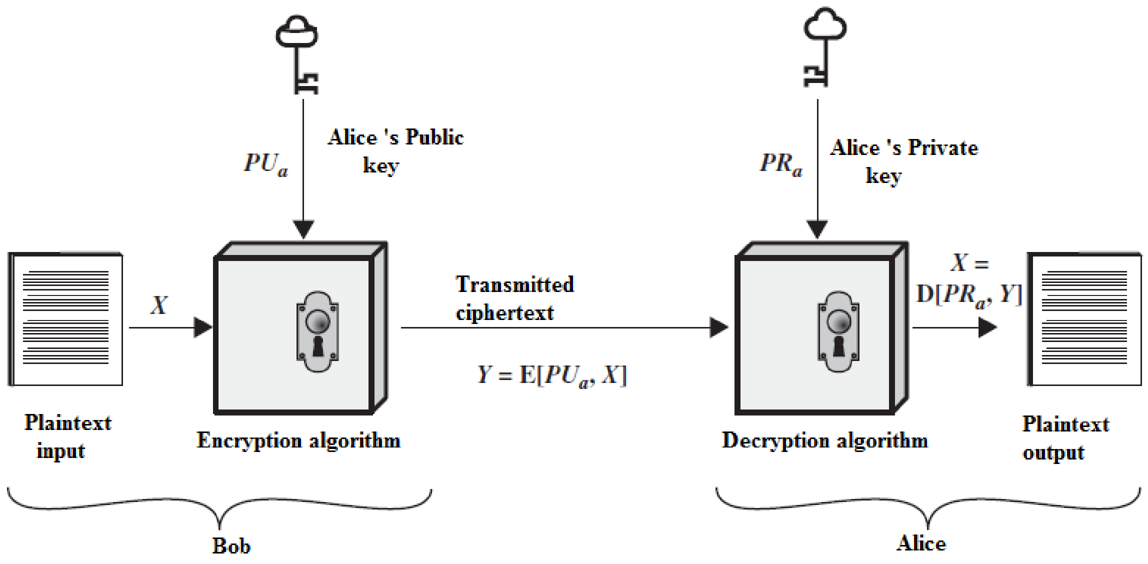


Figure 2.2: Encryption with public key [23]

2.2.2 Block Cipher Operation

A block cipher is an encryption/decryption scheme in which a block of plaintext is treated as a whole and used to produce a cipher text block of equal length. Typically, a block of 64 or 128 bits is used. Many of network –based symmetric cryptographic applications uses block ciphers and these applications have a feistel structure. Feistel structure consists of a number of identical rounds of processing. In each round, a substitution is performed on one half of the data being processed, followed by a permutation that interchanges the two halves. The original key is expanded so that a different key is used for each round. The Data Encryption Standard (DES) has been the most widely used encryption algorithm until recently. It exhibits the classic feistel structure. DES uses a 64-bit block and a 56-bit key. Far more effort has gone into analyzing block ciphers. In general, they seem applicable to a broader range of applications than stream ciphers. The vast majority of network-based symmetric cryptographic applications make use of block cipher.

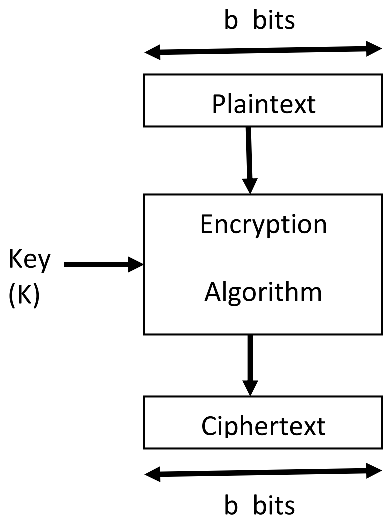


Figure 2.3: Block Cipher

2.2.3 The Data Encryption Standard (DES)

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption. The overall scheme for DES encryption is illustrated in Figure 2.4 As with any encryption scheme; there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length [8].

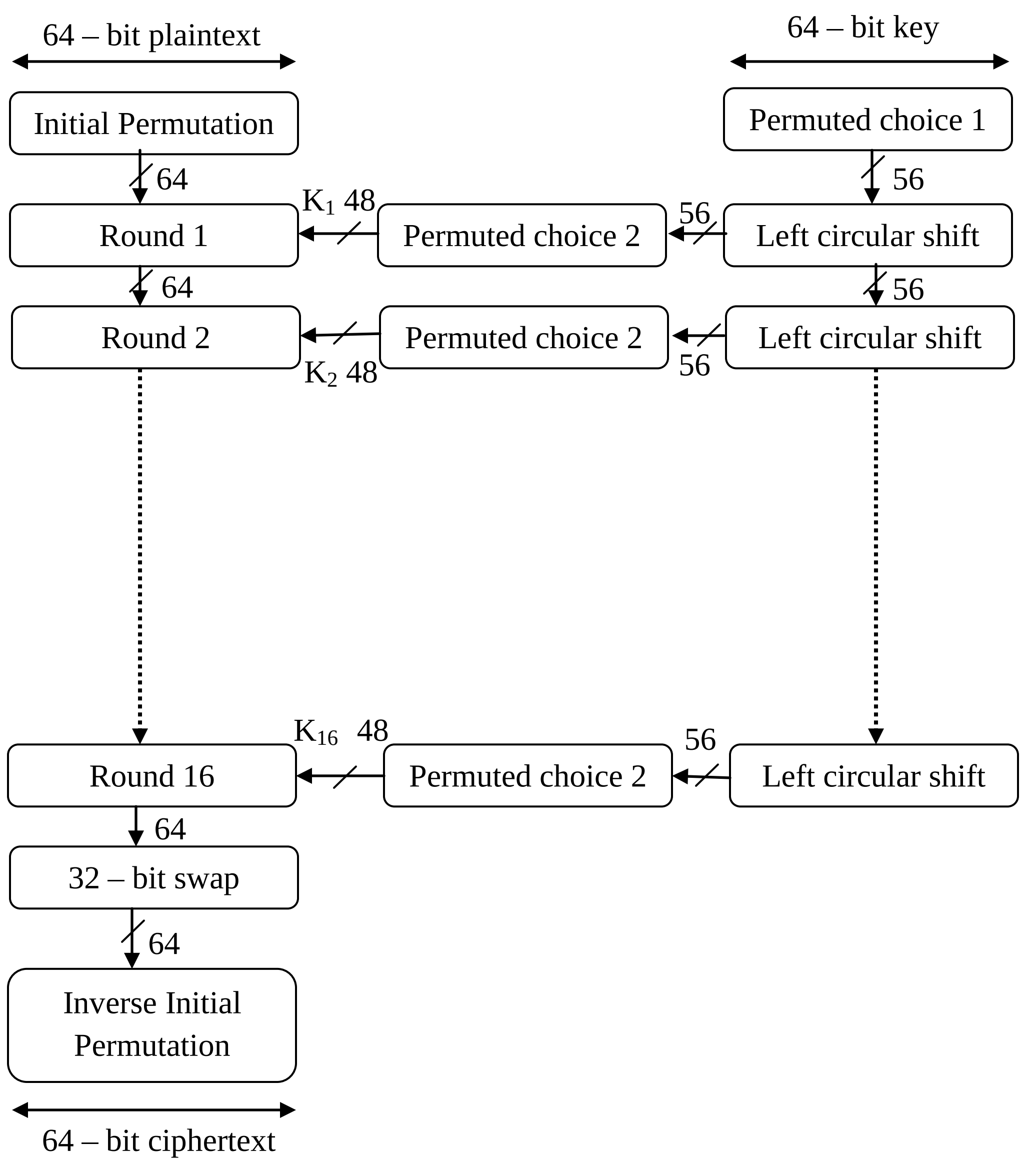


Figure 2.4: General Depiction of DES Encryption Algorithm [23]

The processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*. This is followed by a phase consisting of sixteen rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the preoutput. Finally, the preoutput is passed through a permutation that is the inverse of the initial permutation function, to produce the 64-bit ciphertext.With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher, as shown in Figure 2.4. The right-hand portion of Figure 2.4 shows the way in which the 56-bit key is used. Initially, the key is passed through a permutation function. Then, for each of the sixteen rounds, a subkey (K_i) is produced by the combination of a left circular shift and a permutation. The permutation function is the same for each round, but a different sub key is produced because of the repeated shifts of the key bits [8, 23, and 24].

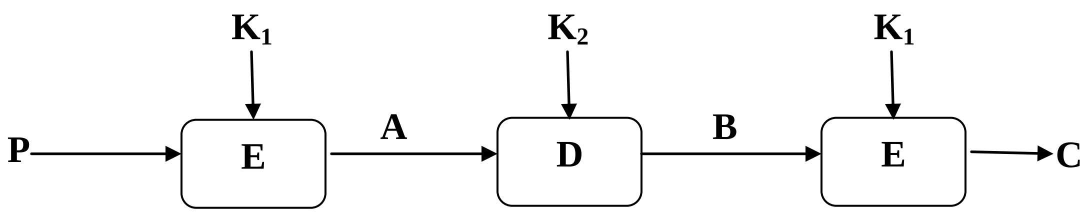
2.2.4. Triple – DES with Two Keys

It is used for encrypting message as same like DES but it requires two keys and takes three stage of encryption with three different keys. It has a drawback of requiring a key length of 168 bits (three keys), which may be unwieldy. As an alternative approach have proposed by Tuchman that uses only two key [23]. The function follows an encrypt – decrypt – encrypt as figure 2.5 [23].

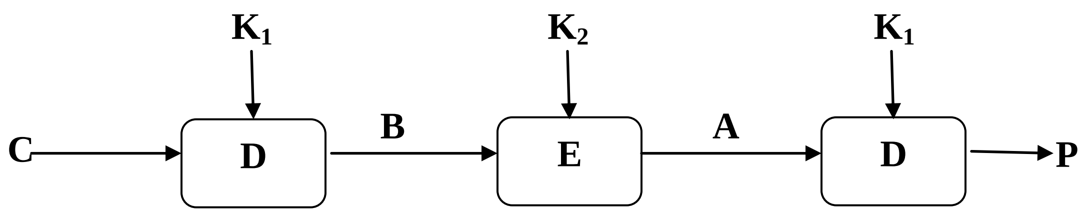
$C = E(K_1 , D(K_2 , E(K_1 , P)))$

$P = D(K_1 , E(K_2 , D(K_1 , C)))$

Here C represents block of ciphertext, P represents plaintext block of message E and D are function of encryption and decryption respectively.



(a) Encryption



(b) Decryption

Figure 2.5. Triple DES with Two Keys

2.2.5. The Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) was published by the National Institute of Standards and Technology (NIST) in 2001. AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. Compared to public-key ciphers such as RSA, the structure of AES and most symmetric ciphers is quite complex and cannot be explained as easily as many other cryptographic algorithms. Figure 2.6 shows the overall structure of the AES encryption process. The cipher takes a plaintext block size of 128 bits, or 16 bytes. The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block 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 2.6. Similarly, the key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words. Figure 2.6 shows the expansion for the 128-bit key. Each word is four bytes, and the total key schedule is 44 words for the 128-bit key. Note that 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. The cipher consists of rounds, where the number of rounds depends on the key length: 10 rounds for a 16-byte key, 12 rounds for 24-byte key and 14 rounds for a 32-byte key. The first rounds consist of four distinct transformation functions: SubBytes, ShiftRows, MixColumns, and AddRoundKey, which are described subsequently. The final round contains only three transformations, and there is a initial single transformation (AddRoundKey) before the first round, which can be considered Round 0. Each transformation takes one or more matrices as input and produces a matrix as output. Figure 2.6 shows that the output of each round is a matrix, with the output of the final round being the ciphertext. Also, the key expansion function generates round keys, each of which is a distinct matrix. Each round key serve as one of the inputs to the AddRoundKey transformation in each round [6,8,23].

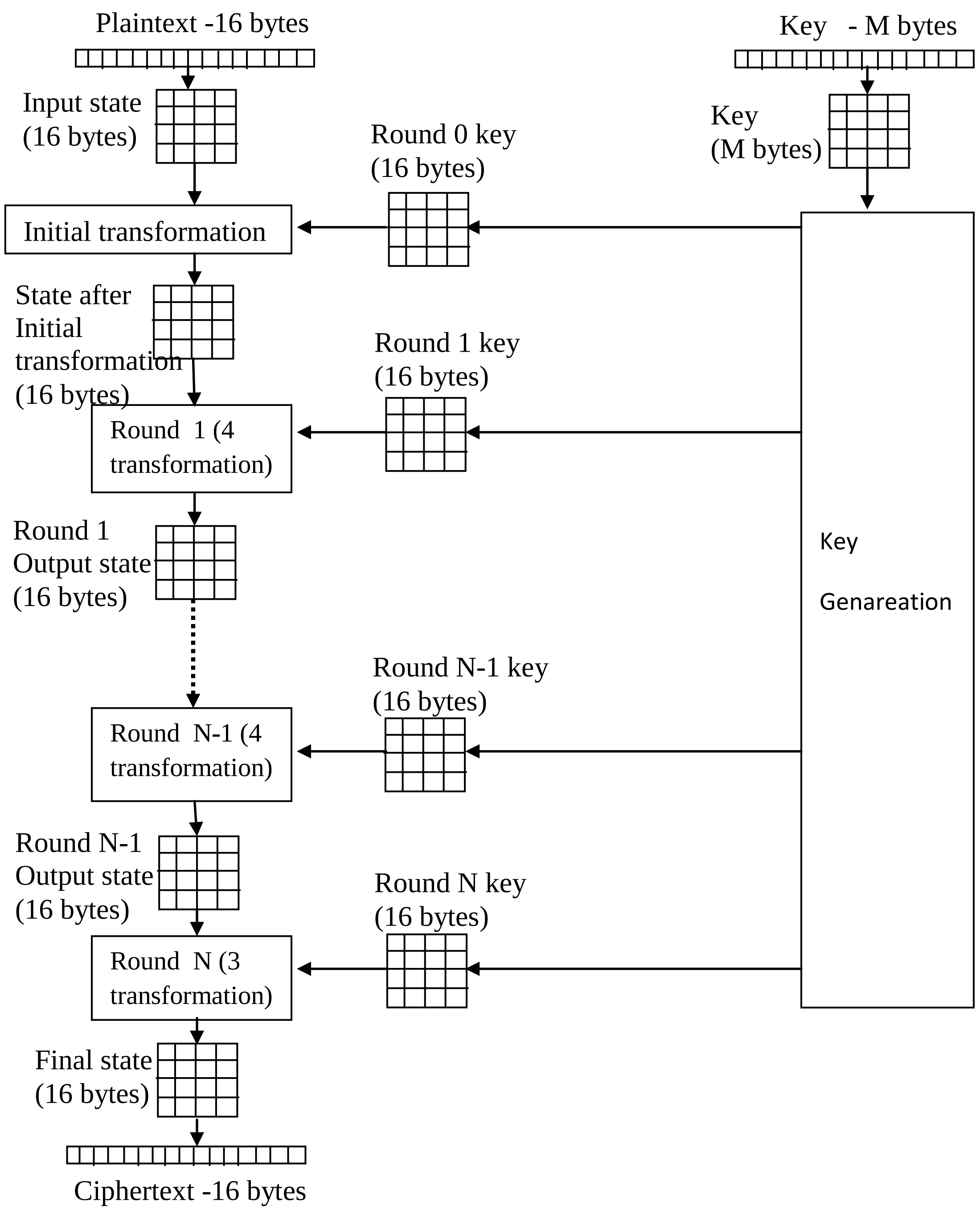
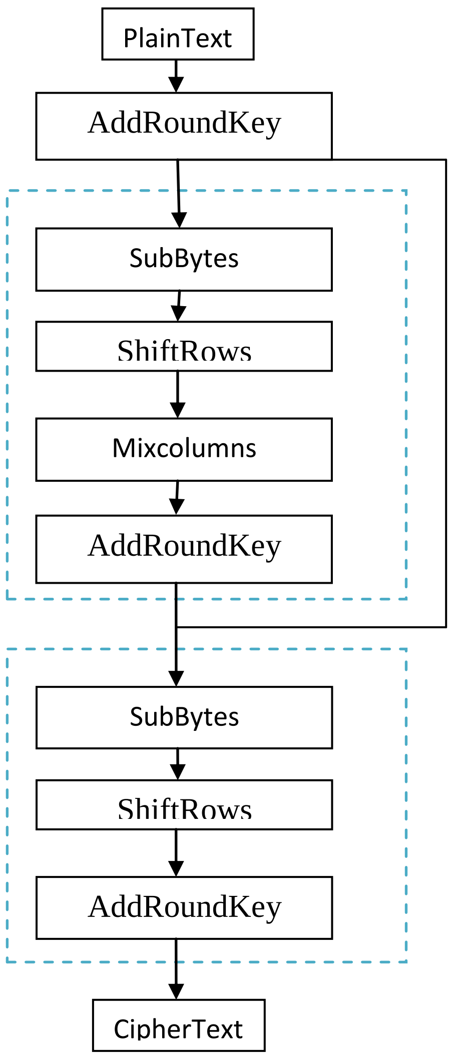
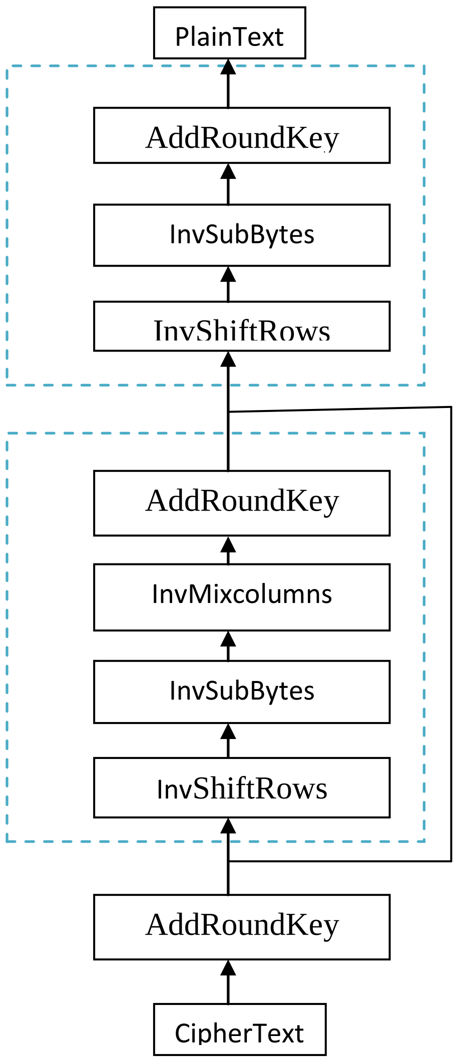


Figure 2.6 : AES Encryption Process[23]



a. Encryption



b. Decryption

Figure 2.7: AES Encryption and Decryption [23]

Chapter 3

Literature Review

In the current scenario, most of the researchers have proposed different message authentication code algorithms to make more secure and fast. Some of them are parallelizable some are using secure hash function like SHA- 1, SHA -2. Several research works to prove the security bounds for each variants of CBC MAC. Some of those literatures have analyzed MAC as pseudorandom function (PRF) similarly Pseudo-Random Functions and Parallelizable Modes of Operations of a Block Cipher have been proposed [13]. Most of researchers have proposed improved security bounds in parallelizable environment as PMAC and non-parallelizable cipher bock based TMAC and XCBC [15].

Many studies have been done on MAC built using block cipher including CBC MAC and its variants. They have been studied the security bounds for each variants of CBC. Performance analysis is done by comparing their key size and number of encryption invocation needed for creating MAC. Number of key scheduled for each algorithm is compared with their security proposed.

3.1 Message Authentication Code (MAC)

More commonly, message authentication is achieved using a message authentication code (MAC), also known as a keyed hash function. Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties. A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC. This can then be transmitted with or stored with the protected message. If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the stored MAC value. An attacker who alters the message will be unable to alter the MAC value without knowledge of the secret key. Note that the verifying party also knows who the sending party is because no one else knows the secret key. In practice, specific MAC algorithms are designed that are generally more efficient than an encryption algorithm [15].

An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a cryptographic checksum or MAC that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key. When A has a message to send to B; it calculates the MAC as a function of the message and the key:

MAC = MAC (K, M)   Where

M = input message

C = MAC function

K = shared secret key

MAC = message authentication code

The message plus MAC are transmitted to the intended recipient. The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC (Figure 3.1). If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.
2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
3. If the message includes a sequence number (such as is used with HDLC, X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number [23].

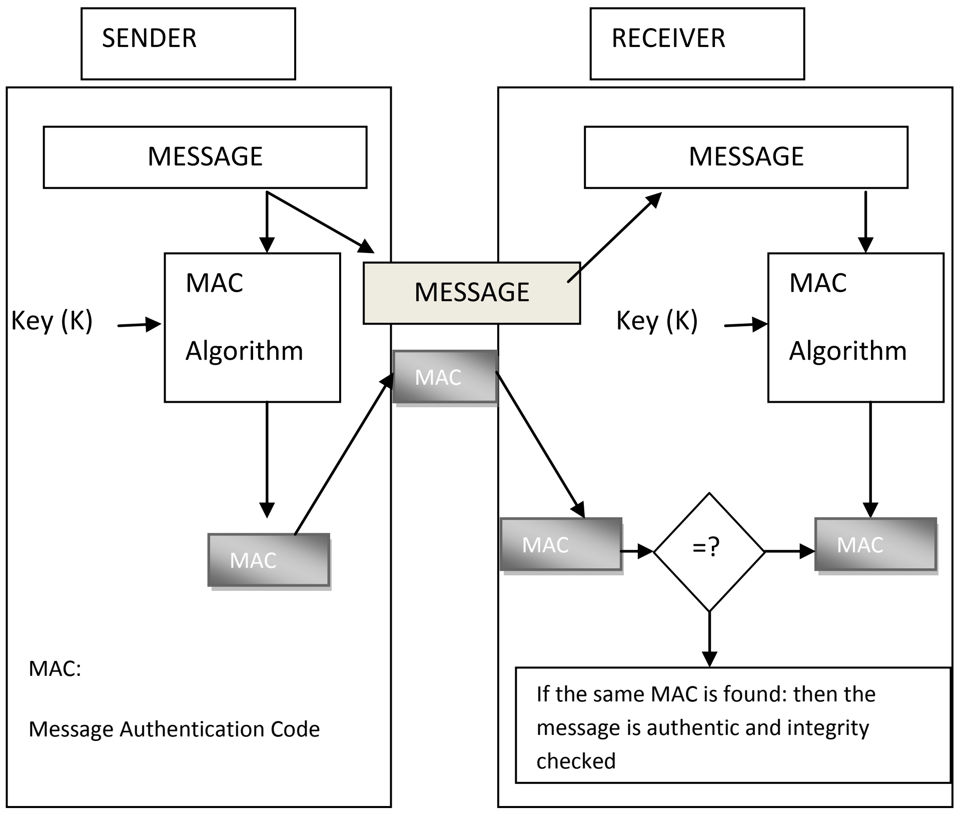

SENDER
RECEIVER
MESSAGE
MESSAGE
MESSAGE
MAC Algorithm
MAC Algorithm
Key (K)
Key (K)
MAC
MAC
MAC
MAC
=?
MAC:
Message Authentication Code
If the same MAC is found: then the message is authentic and integrity checked

Figure 3.1: Message Authentication Code

3.2. The Field with 2^n Points

A point a in GF(2^n) can be viewed as the following ways[12]:

- As an abstract point in a field;
- As an n – bit string a_{n-1} …………$a_1a_0 \in \{0,1\}^n$;
- As a formal polynomial $a(u) = a_{n-1}\ u^{n-1} + \ldots\ldots + a_1u + a_0$ with binary coefficients.

To add two points in GF (2^n), takes their bitwise XOR and denoted by $a \oplus b$. But to multiply two points, fix some irreducible polynomial f(u) having binary coefficients and degree n. some standard irreducible polynomials for different algorithm which are used GF(2^n) multiplication operation.

$f(u) = u^{64} + u^4 + u^3 + u + 1$ for n = 64

$f(u) = u^{128} + u^7 + u^2 + u + 1$ for n = 128

First polynomial is used in CBC MAC variants when TDES algorithm is used as encryption algorithm to derive key as L.u and $L.u^2$ and second polynomial is used in AES.

It is particularly easy to multiply a point $a \in \{1, 0\}^n$ by u . For example

If n = 128, where $f(u) = u^{128} + u^7 + u^2 + u + 1$, then multiplying $a = a_{127} \ldots\ldots\ldots a_1 a_0$ is done by following ways:

$$a.u = \begin{cases} a \ll 1 & \text{if } a_{127} = 0 \\ (a \ll 1) \oplus 0^{120}10000111 & \text{otherwise} \end{cases}$$

Where a.u = a (u) .u mod f (u).

If n = 64, where $u^{64} + u^4 + u^3 + u + 1$. Then multiplying $a = a_{63} \ldots\ldots\ldots a_1 a_0$ is done by following ways:

$$a.u = \begin{cases} a \ll 1 & \text{if } a_{63} = 0 \\ (a \ll 1) \oplus 0^{59}11011 & \text{otherwise} \end{cases}$$

3.3 Key Generation

The key length of CBC MAC is k bit only where k is the bit length of secrete key share by sender and receiver but key length for other variants are vary than k bit. Where EMAC required 2k bit obtained from two secrete key. XCBC required three key one k bit and other two key have 2n bits, here n is the length of block size of message. Suggested by [11] it is reduced as following :

K1 = the first k bits of $E_k(C_{1a}) \| E_k(C_{1b})$,

K2 = $E(C_2)$,

K3 = $E(C_3)$. Where C_{1a} , C_{1b}, C_2, C_3 are distinct constants.

Next two variants TMAC and OMAC are the refinement of XCBC. Key for TMAC is obtained by replacing $K_2.u$ and keys are generated in OMAC by replacing K_2 as L.u and K_3 by $L.u^2$.

Where u be the some positive constant and "." is a multiplication in GF(2^n) and L is obtained as follows:

L = $E_k(0^n)$[12].

3.4 Cipher Block Chaining (CBC)

CBC is a technique in which the same plaintext block, if repeated, produces different ciphertext blocks. In this scheme, the input to the encryption algorithm is the XOR of the current plaintext block and the preceding ciphertext block; the same symmetric key is used for each block. In effect, it is chained together the processing of the sequence of plaintext blocks. The input to the encryption function for each plaintext block bears no fixed relationship to the plaintext block.Therefore, repeating patterns of bits are not exposed. As with the ECB mode, the CBC mode requires that the last block be padded to full bits if it is a partial block. For decryption, each cipher block is passed through the decryption algorithm. The result is XORed with the preceding ciphertext block to produce the plaintext block [2,8]. It can be viewed as

$C_j = E(K, [C_{j-1} \oplus P_j])$

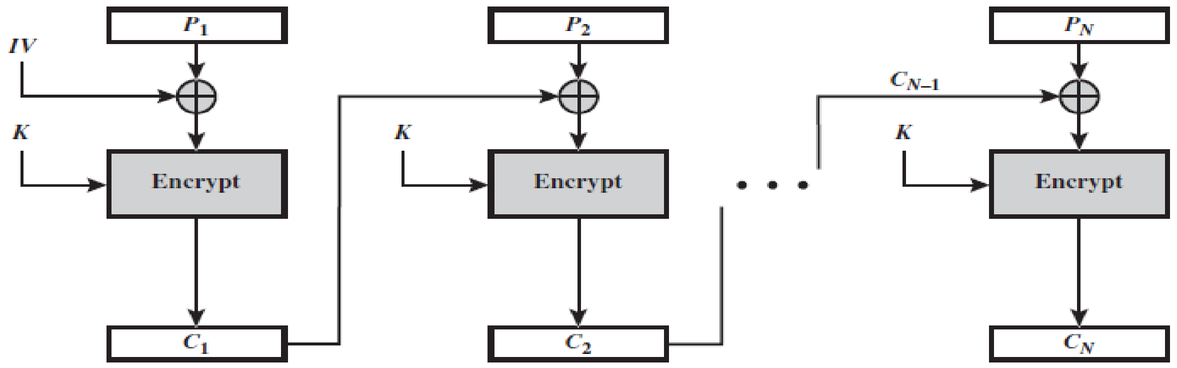


Figure 3.2(a): Encryption of CBC Mode [21]

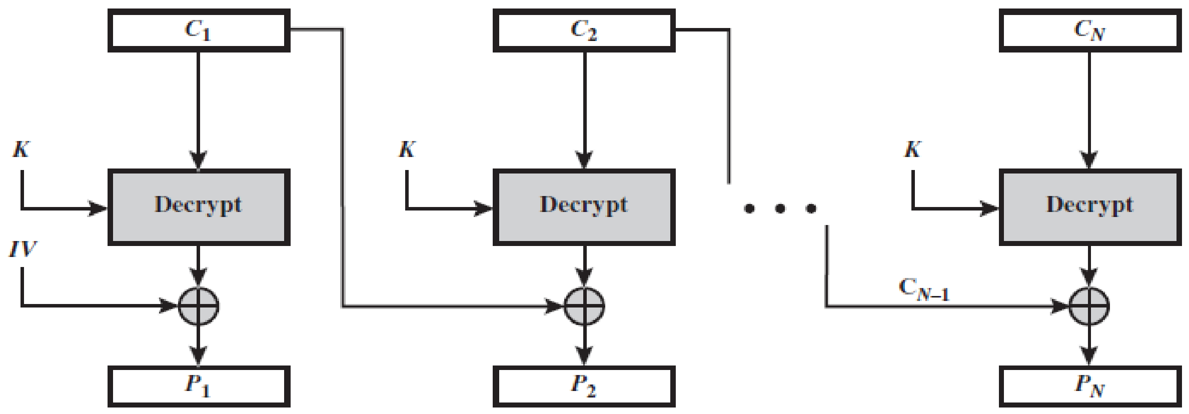


Figure 3.2(b): Decryption of CBC mode

3.4.1. Cipher Block Chaining Message Authentication Code (CBC -MAC)

CBC-MAC is a technique for constructing a message authentication code using a block cipher. The message is divided into equal number of block and encrypted with some block cipher algorithm in CBC mode to create a chain of blocks such that each block depends on the proper encryption of the previous block. This interdependence ensures that change to any of the plaintext bits causes the final encrypted block to change in a way that cannot be predicted without knowing the key to the block cipher. M can be broken into blocks such that

$M = M_1, M_2 \ldots M_m$,where M_i represents block of message.Then each block is passed through the encryption E with key K and the result is XORed with the next block. If E_K represents the encryption using a secrete key K then cipher block chaining is given as follows algorithm[2, 12]:

Algorithm

INPUT: block of message (M)

OUTPUT: MAC (Tag)

```
Algorithm CBC-MAC_K (M)
Partition M into M [1]………M[m]
C [0] = $0^n$
for i = 1 to m do
C[i] = $E_K(C[i-1] \oplus M[i])$
Tag = C[m]
return Tag
```

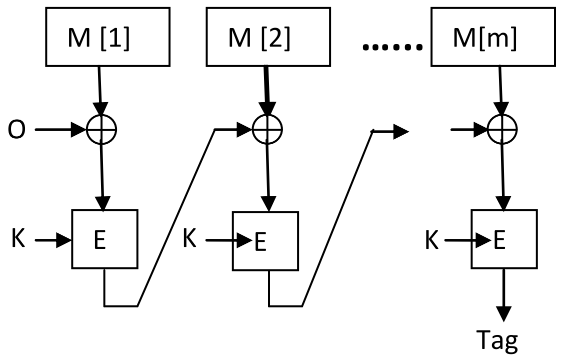


Figure 3.3: Illustration of CBC -MAC

The CBC-MAC comes in different versions varying in details such as padding, length variability. The general way of padding for CBC-MAC is by considering the final input block as a partial block of data, left justified with zeroes appended to form a full block.

3.4.2 Encrypted –MAC (EMAC)

To deal with variable message length in blocks m, Encrypted MAC (EMAC) was developed. MAC encrypts $\mathrm{CBC}_{E_{K_1}}(M)$ using a new block cipher key K_2 [12]. That is:

$$\mathrm{EMAC}_{E_{K_1},E_{K_2}}(M) = E_{K_2}\left(\mathrm{CBC}_{E_{K_1}}(M)\right).$$

The EMAC is a popular variant of the CBC-MAC which was developed by the RACE project. A problem is that the message length is limited to a positive multiple of n, that is, the domain is limited to $(\{0, 1\}^{n})^{+}$. The simplest approach to deal with message whose lengths are not a multiple of n is append the minimal 10^{i} to M as a padding so that the length is multiple of n. Note that the padding is appended even if the size of the message is already a multiple of n [11]. EMAC can deal with completely variable message length, that is, the domain is $\{0,1\}^{*}$.

Algorithm

INPUT: block of message (M)

OUTPUT: MAC (Tag)

Algorithm $\mathrm{EMAC}_{K1, K2}$ (M)

Partition M into M [1]………M[m]

```
C [0] = $0^n$
for i = 1 to m do
C[i] = $E_{K1}$(C [i-1] $\oplus$ M[i])
Tag = $E_{K2}$ ($E_{K1}$(C [m-1] $\oplus$ M[m]))
return Tag
```

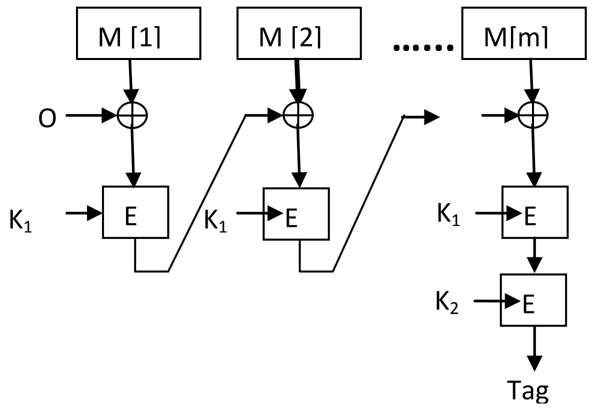


Figure 3.4 : Illustration of EMAC

3.4.3 XCBC MAC

XCBC takes three key: one block cipher key K1, and two n –bit keys K2 and K3. XCBC makes two cases to deal with arbitrary length messages: $M \in (\{0, 1\})^+$ and $M \notin (\{0, 1\})^+$. If $M \in (\{0, 1\})^+$ then XCBC computes exactly the same as CBC MAC, except XORing an n-bit key K_2 before encrypting the last block. If $M \notin (\{0, 1\})^+$ then minimal 10^i padding ($i \geq 0$) is appended to M so that the length is a multiple of n, and XCBC computes exactly the same as the CBC MAC, except XORing another n-bit key K3 before encrypting the last block.[2,12]

Algorithm

INPUT: block of message (M)

OUTPUT: MAC (Tag)

```
Algorithm XCBCMAC$_{K1, K2, K3}$ (M)
Partition M into M [1]………M[m]
C [0] = $0^n$
for i = 0 to m-1 do
C[i] = $E_{K1}$(C [i-1] $\oplus$ M[i])
```

```
if | M[m] | = n then Tag = $E_{K1}$ (C [m-1] $\oplus$ M[m] $\oplus$ K2)
            else Tag = $E_{K1}$ (C [m-1] $\oplus$ M[m] 10....0 $\oplus$ K3)
return Tag
```

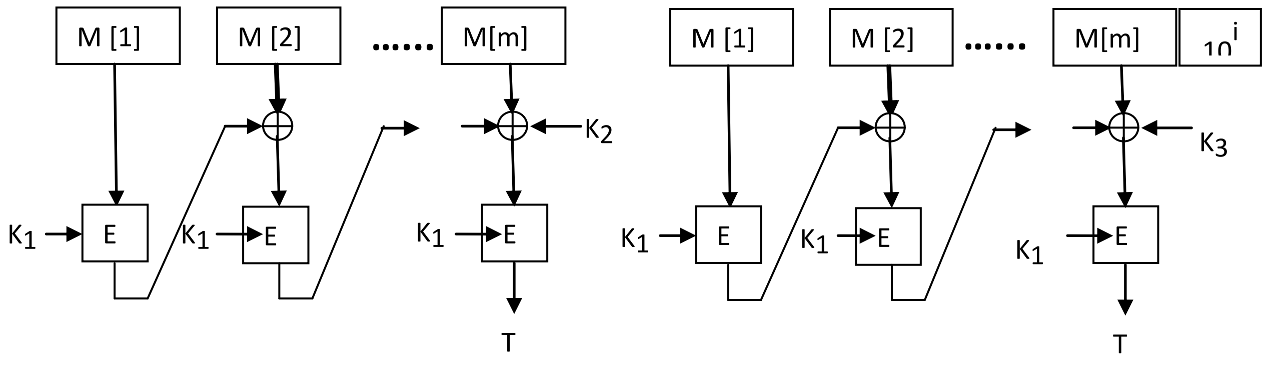


Figure 3.5: Illustration of XCBC

3.4.4 TMAC

Two key CBC Message authentication code is a refinement of XCBC. It uses only (k+n)bit key for TMAC while XCBC uses (k+2n) bit key ,where k is the key length of the underlying block cipher and n is its block length. TMAC is obtained from XCBC by replacing (*K*2,*K*3) with (*K*2.u,*K*2), where u is some non-zero constant and "·" denotes multiplication in GF(2*n*).[12,24]

Algorithm

INPUT: block of message (M)

OUTPUT: MAC (Tag)

```
Algorithm TMAC$_{K1, K2}$ (M)
Partition M into M [1]………M[m]
C [0] = $0^n$
for i = 0 to m-1 do
C[i] = $E_{K1}$(C [i-1] $\oplus$ M[i])
if | M[m] | = n then Tag = $E_{K1}$ (C [m-1] $\oplus$ M[m] $\oplus$ K2.u)
            else Tag = $E_{K1}$ (C [m-1] $\oplus$ M[m] 10....0 $\oplus$ K2)
return Tag
```

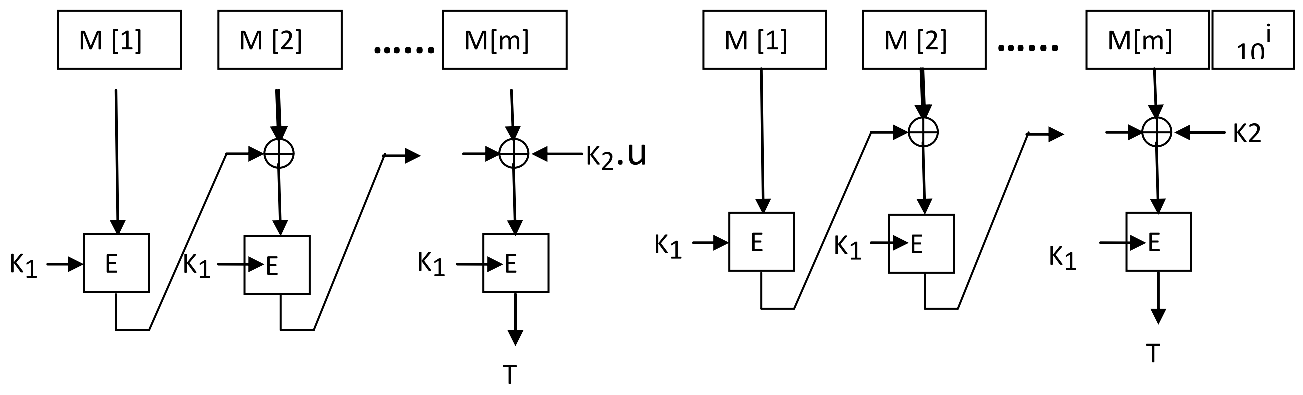


Figure 3.6 : Illustration of TMAC

3.4.5 OMAC 1

One-key CBC MAC takes only one key,K (k bits) of a block cipher E. Previously ,XCBC requires three keys,(k+2n) bits in total, and TMAC requires two keys,(k+n) bits in total, where n denotes the block length of E. OMAC is a generic name of OMAC1 and OMAC2. OMAC1 is obtained from XCBC by replacing (K_2, K3) with (L.u, L.u2) for some non-zero constant u in GF (2^n), where L is given by [2, 9]

$$L = E_K (0^n).$$

OMAC2 is similarly obtained by using (L.u, $L.u^{-1}$).

Algorithm

INPUT: block of message (M)

OUTPUT: MAC (Tag)

```
Algorithm OMAC-family$_K$(M)
  L = $E_K$(Cst)
Y[0] = $0^n$
Partition M into M [1]………M[m]
for i = 0 to m-1 do
Y[i] = $E_{K1}$(Y [i-1] $\oplus$ M[i])
X[m] = $pad_n$(M[m] $\oplus$ Y [m-1] )
if | M[m] | = n then X[m] = X[m] $\oplus$ L.u
                else X[m] = X[m] $\oplus$ $L.u^2$
T = $E_K$(X[m])
return T
```

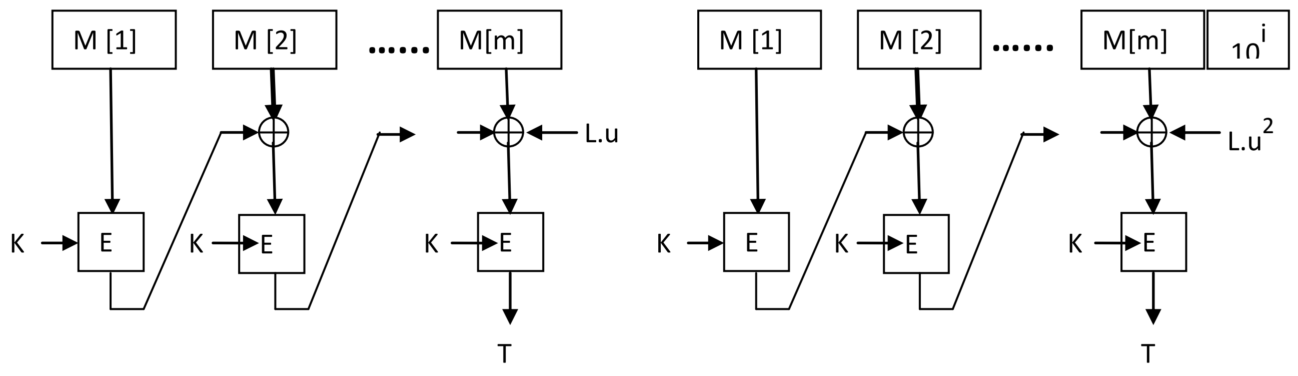


Figure 3.7: Illustration of OMAC1

Chapter 4

Java Implementation

Java was conceived at Sun Microsystems, in 1991. This language is initially called “OAK” but it was renamed as java in 1995 with the Virtual Machine being known as the Java Virtual Machine (JVM). At that time, the use of the World Wide Web was starting to become widespread. The web involved the communication between all sorts of processors and systems; just the sort of situation for which Sun had developed Java. Hence Java became the preferred language for Web programming [20].
Java compiles the source file (.java) and converts into intermediate file called byte code (.class) which can be run on several architecture with the help of java virtual machine (JVM).this beauty of the java programming language motivates to use of java anywhere or in any type of application development. This makes software developed in java platform independent.

4.1 Choice of the Programming Language: Java

Most of other language likes C, C++ are designed to be compiled for a specific target machine. Although it is possible to compile a C++ program for any type of CPU, to do so requires a full C++ compiler targeted for that CPU. The problem is that compilers are expensive and time consuming to create, solution was needed, and to find a solution, java was created which could be used to produce code that can run on a variety of CPUs under different environment.
The Java security APIs spans a wide range of areas. For developing secure application, Cryptographic and public key infrastructure (PKI) interfaces are used. Interfaces for performing authentication and access control enable applications to protect against unauthorized access to protected resources. The APIs allow for multiple interoperable implementations of algorithms and other security services. Services are implemented in providers, which are plugged into the Java platform via a standard interface that makes it easy for applications to obtain security services without having to know anything about their implementations. This allows developers to

focus on how to integrate security into their applications, rather than on how to actually implement complex security mechanisms. The Java platform includes a number of providers that implement a core set of security services. It also allows for additional custom providers to be installed. This enables developers to extend the platform with new security mechanisms [24].

4.2 Netbeans

NetBeans is an integrated development environment (IDE) for developing primarily with Java, but also with other languages, in particular PHP, C/C++, and HTML5. It is developed at Charles University as a student project in 1996. In 1997, it was produced as commercial versions and bought by Sun Microsystems in 1999.
It is also an application platform framework for Java desktop applications and others. The NetBeans IDE is written in Java and can run on Windows, OS X, Linux, Solaris and other platforms supporting a compatible JVM. The NetBeans Platform allows applications to be developed from a set of modular software components called modules. Integrated Development Environment (IDE) that is often the front end of a programming system.
Different versions of Netbeans IDE are introduced in last few years. NetBeans IDE 7.0 was released in April 2011. On August 1, 2011, the NetBeans Team released NetBeans IDE 7.0.1, which has full support for the official release of the Java SE 7 platform. As passing versions from NetBeans IDE 6.5 to currently developing version NetBeans IDE 8.0 many more features are added in newer versions. NetBeans IDE 7.4 was released in October 15, 2013.NetBeans IDE 8.0 is currently in development. NetBeans IDE 7.0.1 is used for implementing in this thesis [16, 24].

4.3 Implementation Details of Candidate algorithm

The four variant of Cipher block chaining message authentication code are implemented along with the CBC MAC. The calling procedure of all encryption algorithms are same and various size of input file is given to each of the variants. For creating large file of message as string following code is used:

```
String str = "";
  try
      {
         FileReader fr = new FileReader(file+".txt");
         int c;
         while ((c = fr.read()) != -1)
         {
            str += (char) c;
         }
      }
      catch (IOException e)
      {

         System.out.println("I/O Error: " + e);

     }
```

The file is supplied as input to every candidate algorithms as string, which is converted into byte form using the following code:

```
Byte[]  input = str.getBytes();
```

Each size of file is converted into byte form and finally binary String is formed from byte form of input message, which is created by following code:

```
String binaryStringRep ="";

        byte[] bytes = str.getBytes();

        for(int i = 0; i < bytes.length; i++)

        {

  binaryStringRep += String.format("%8s", Integer.toBinaryString(bytes[i] & 0xFF)).replace('
', '0');        }
```

The binary string formed by above code section is checked either it is actually integer multiple of block size or not where TDES needs 64 bits and AES needs 128 bits as each block of message for being input to encryption algorithm. Before encrypting each block of message there must be performed XOR operation with previous block of cipher text which is done by following procedure in this thesis:

```
 public byte[] xOR(byte[] currentM, byte[] C) throws
UnsupportedEncodingException    {

       String currentMbinary="";
       String  Cbinary="";
       for(int i = 0; i < currentM.length; i++)         {
             currentMbinary += String.format("%8s",
Integer.toBinaryString(currentM[i] & 0xFF)).replace(' ', '0');
             Cbinary += String.format("%8s", Integer.toBinaryString(C[i] &
0xFF)).replace(' ', '0');
          }
       String xorstr ="";
     for(int i = 0 ;i<128; i++)     {
       if(currentMbinary.charAt(i) == Cbinary.charAt(i))       {

         xorstr += "0";

       }       else       {

          xorstr += "1";

       }   }

       String mstr = "";

           for(int j = 0; j <= 128 - 8; j+=8)  {

                  int k = Integer.parseInt(xorstr.substring(j, j+8), 2);

                  mstr += (char) k;

                }             byte xorbyte [] = mstr.getBytes();

           return xorbyte;  }
```

If the Encryption algorithm is TDES, Encryption is performed as the following procedure:

```
public byte[] encrypTDES(SecretKey secretKeyTDES_K1,SecretKey
secretKeyTDES_K2,byte[] message) throws Exception
   {
      Cipher cipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
      cipher.init(Cipher.ENCRYPT_MODE, secretKeyTDES_K1);
      byte[] cipherText = cipher.doFinal(message);

      Cipher decipher = Cipher.getInstance("DES/ECB/noPadding");
      decipher.init(Cipher.DECRYPT_MODE, secretKeyTDES_K2);
      byte[] plainText = decipher.doFinal(cipherText);

      Cipher cipher1 = Cipher.getInstance("DES/ECB/PKCS5Padding");
      cipher1.init(Cipher.ENCRYPT_MODE, secretKeyTDES_K1);
      byte[] cipherTextfinal = cipher1.doFinal(plainText);

      return cipherTextfinal;
   }
```

If the Encryption algorithm is AES, Encryption is performed as the following procedure:

```
   public byte[] encryptAES(SecretKey secretKeyAES,byte[] plainTextBytes)
throws Exception

   {

     Cipher  cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

      cipher.init(Cipher.ENCRYPT_MODE, secretKeyAES);

      final byte[] cipherText = cipher.doFinal(plainTextBytes);

      return cipherText;

   }
```

4.3.1 CBC MAC

CBC MAC is the first message authentication technique based on cipher block chaining mode. It is secured only when the message size is multiple of block size of encryption algorithm. Causes of insecurity of CBC MAC, other variants are introduced. Every variants of CBC MAC algorithm must have scheduled symmetric key of different bit size like 112 bits for TDES and 128 bits for AES before starting the encryption of first block of message which is obtained by the following procedure.

```
SecretKey AES_keyGeneration()
    {
        SecretKey secretKey = null;
         try {
    KeyGenerator keygenerator = KeyGenerator.getInstance("AES");
    secretKey = keygenerator.generateKey();
     }
      catch(Exception e)
      {
       JOptionPane.showMessageDialog(null,e.getStackTrace());
      }
         return secretKey;
    }
```

Where KeyGenerator is the class in javax.crypto package and SecretKey in javax.crypto package. For key generation of TDES same code can be used instead of giving parameter of getInstance() method as “DES”. Generated key is supplied to the encryption algorithm like AES as following code section:

```
    String IV = "0000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
000";
String ivstr=”;
for(int i = 0; i <= 128 - 8; i+=8){
   int k = Integer.parseInt(IV.substring(i, i+8), 2);
   ivstr += (char) k;
}
        byte [] IVbyte = ivstr.getBytes();
        String binaryStringRep ="";
        byte[] bytes = str.getBytes();
        for(int i = 0; i < bytes.length; i++)  {
          binaryStringStringRep += String.format("%8s", Integer.toBinaryString(bytes[i] &
0xFF)).replace(' ', '0');
        }
      if(binaryStringRep.length()%128==0)  {
        for(int i = 0 ;i<binaryStringRep.length()-128 ;i+=128)  {
          String bmstr = binaryStringRep.substring(i,i+128);
          String mstr = "";
         for(int j = 0; j <= 128 - 8; j+=8)    {
                int k = Integer.parseInt(bmstr.substring(j, j+8), 2);
                mstr += (char) k;
              }
          byte mbyte [] = mstr.getBytes();
          byte [] xordm  = xOR(mbyte,IVbyte);
           String ss ="";
        for(int f = 0; f < xordm.length; f++){
           ss += String.format("%8s", Integer.toBinaryString(xordm[f] & 0xFF)).replace(' ',
'0');}
     String sss  = ss.substring(0,128);
        String mstr1 = "";
          for(int j = 0; j <= 128 - 8; j+=8){
                int k = Integer.parseInt(sss.substring(j, j+8), 2);
                mstr1 += (char) k;  }
          byte m [] = mstr1.getBytes();
          byte [] nextcipher = new Process().encryptAES(secretKey
secretKeyTDES_EMAC_k1  =  p.DES_keyGeneration();
      secretKeyTDES_EMAC_k11 = p.DES_keyGeneration();
      secretKeyTDES_EMAC_K2  = p.DES_keyGeneration();
      secretKeyTDES_EMAC_K22=p.DES_keyGeneration();  AES_OMAC_K1,m);
          IVbyte = nextcipher;
                }
```

4.3.2 EMAC

EMAC needs one more encryption key than CBC MAC therefore two individual key are generated by the same process. In this work triple DES with two key is used so for EMAC implementation, four random secrete key are generated first two keys are used to encrypt every cipher in chain but other two keys are used to encrypt MAC again to get final Tag(MAC). But for AES encryption algorithm only two 128 bit keys are produced by following code segment:

```
secretKeyTDES_EMAC_k1  =  p.DES_keyGeneration();

 secretKeyTDES_EMAC_k11 = p.DES_keyGeneration();

 secretKeyTDES_EMAC_K2  = p.DES_keyGeneration();

 secretKeyTDES_EMAC_K22=p.DES_keyGeneration();
```

Where DES_keyGeneration() is a procedure to generate secrete key as already mention in above section which return 64 bit secrete key for DES encryption algorithm. Similarly for AES:

```
secretKeyAES_EMAC_k1  = p.AES_keyGeneration();

secretKeyAES_EMAC_k2 = p.AES_keyGeneration();
```

4.3.3 XCBC

XCBC requires three key but for reduce key length as [11] suggest three key may be generated by using only one symmetric encryption algorithm but to make it possible four constant values are needed in my study, these values are set up as C_{1a} = 78945612 , C_{1b} = 12345678 , C_2 = 15935778 and C_3 = 12357896

```
byte [] first  = p.encrypTDES(secretKeyTDES_K1,secretKeyTDES_K11,"78945612".getBytes());

            byte [] second =
p.encrypTDES(secretKeyTDES_K1,secretKeyTDES_K11,"12345678".getBytes());

            byte [] k1 = (first.toString() + second.toString()).getBytes();//56 bit

            String binaryStringStringRep ="";

        for(int i = 0; i < k1.length; i++)  {

          binaryStringStringRep += String.format("%8s", Integer.toBinaryString(k1[i] &
0xFF)).replace(' ', '0');

        }

        String s =binaryStringStringRep.substring(0,64);

             String mstr = "";

         for(int j = 0; j <= 64 - 8; j+=8) {

               int k = Integer.parseInt(s.substring(j, j+8), 2);

               mstr += (char) k;

             }

          byte mbyte [] = mstr.getBytes();

             XCBC_TDES_K1 = new SecretKeySpec(mbyte, "DES");

             XCBC_TDES_K11 = p.DES_keyGeneration();

             XCBC_TDES_K2 = p.encrypTDES(secretKeyTDES_K1,secretKeyTDES_K11,
"15935778".getBytes());//64bit

             XCBC_TDES_K3 =
p.encrypTDES(secretKeyTDES_K1,secretKeyTDES_K11,"12357896".getBytes());//64bit
```

4.3.4 TMAC

TMAC is implemented in the same way of XCBC because it is the refinement of XCBC where three key needed in XCBC is reduced by one key and became Two –key MAC. Three keys are treated in this algorithm is as second key, say K_2 in previous variants is replaced by K_2. u where "." Is multiplication over GF (2^n). The code segment for converting K_2 to K_2 .u is as follows:

```
 byte  L[]
=p.encrypTDES(secretKeyTDES_K1,secretKeyTDES_K11,"00000000".getBytes();
            String bin ="";
        for(int i = 0; i < L.length; i++)
         {
           bin += String.format("%8s", Integer.toBinaryString(L[i] & 0xFF)).replace(' ',
'0');
         }
          String sss  = bin.substring(0,64);
          String ssss = "";
        if(sss.substring(0,1).equals("0"))  {
       ssss = sss.substring(1,64)+"0";
         }
        else {
             String m = sss.substring(1,64)+"0";
             String m1 =
"0000000000000000000000000000000000000000000000000000000000011011";
             for(int i = 0;i<64;i++)  {
             if(m.charAt(i)==m1.charAt(i))
               ssss+="0";
               else
         ssss+="1";
             }      }
         String str = "";
         for(int j = 0; j <= 64 - 8; j+=8) {
                int k = Integer.parseInt(ssss.substring(j, j+8), 2);
                str += (char) k;
              }
          byte mmbyte [] = str.getBytes();
          k2_OMAC_TDES = mmbyte;
```

4.3.5 OMAC

This is also the refinement variants of CBC MAC. It is generated by replacing three key in XCBC with L.u and L.u^2 instead of $K_{2,}$ and $K_{3.}$ Implementation of this algorithm is similar to TMAC only the following code segment should be added to get third key as XCBC. Third key (K_3) is obtained by L.u$^{2.}$

```
  String ssss1 = "";
        if(sss1.substring(0,1).equals("0"))
        {
      ssss1 = sss1.substring(1,64)+"0";
        }
        else
        {
             String m = sss1.substring(1,64)+"0";
             String m1 =
"0000000000000000000000000000000000000000000000000000000000011011";
             for(int i = 0;i<64;i++)
             {
             if(m.charAt(i)==m1.charAt(i))
             {
               ssss1+="0";
             }
             else
             {
                ssss1+="1";
             }
             }

        }

        String str1 = "";
         for(int j = 0; j <= 64 - 8; j+=8)
             {
                int k = Integer.parseInt(ssss1.substring(j, j+8), 2);
                str1 += (char) k;
              }
          byte mmbyte1 [] = str1.getBytes();

          k3_OMAC_TDES = mmbyte1;
```

Where sss1 is the binary string of Key obtained from L.u and using above code segment ,L.u^2 is obtained to replace K_3 in XCBC Variant.

4.4 Sample Test Cases

For testing data input, the different size of text file is taken as input message. Size of 19 bytes file has taken as smallest size. Key is produced by using library function of java. Input message and generated key is as follows:

4.4.1 Key

I. AES -128

K_1 = 2b7e151628aed2a6abf7158809cf4f3c

II. TDES

$\mathbf{K_1}$ **=** 8aa83bf8cbda1062

$\mathbf{K_2}$ **=** 0bc1bf19fbb6cd58

4.4.2 Input message (29 byte)

“Message authentication code”

4.4.3 Message authentication code(MAC)

CBC TDES	:	C28F006EC299C284
CBC AES	:	C2A8C38BC29335C3A805C2A8C2B7C3B4
EMAC TDES	:	4439C2A822231E41
EMAC AES	:	0BC3BA0CC3AB6F6FC28DC2AA31C390C3
XCBC TDES	:	6BC28EC28EC3BC77C39D1632
XCBC AES	:	C3B85CC29E0873725D7DC2B9C3B5C2A9
TMAC TDES	:	C29E0236C2901146
TMAC AES	:	C3A6C3830D312D05C29EC29522C387C2
OMAC TDE	:	3CC3A8C3952DC387
OMAC AES	:	2BA46C2962DC3AF5CC3B0C282C381504

Where CBC TDES is the message authentication code for CBC MAC using triple- DES as encryption algorithm and similar to other.

4.4.4 Input message (595byte)

The CBC MAC is a well-known method to generate a message authentication code (MAC) based on a block cipher. Bellaire, Kalian, and Roadway proved the security of the CBC MAC for fixed message length mn bits, where n is the block length of the underlying block cipher E. However, it is well known that the CBC MAC is not secure unless the message length is fixed. Therefore, several variants of CBC MAC have been proposed for variable length messages. First Encrypted MAC (EMAC) was proposed It is obtained by encrypting the CBC MAC value by E again with a new key K2.

4.4.5 Message authentication code(MAC)

CBC TDES	:	33C3AE177DC38AC2
CBC AES	:	C3943CC2AD455818C3BA60C39B25C39D
EMAC TDES	:	F58C3B7C38C2206C
EMAC AES	:	C287C3B5C3B26E45C3ADC3ADC29DC390
XCBC TDES	:	0F1BC2A8C3A5C2BA
XCBC AES	:	1BC28E1170C28913C3871223C28EC3BEC5
TMAC TDES	:	C3B92D6100C288C3
TMAC AES	:	070FC29F1547591A48C280C3A658C2A4
OMAC TDE	:	5EC39AC3A2C398C3
OMAC AES	:	30C3BD79C29BC2AE4A7D03C3BBC2B224

Chapter 5

Result and Analysis

This chapter presents an overview of comparison of the CBC MAC and it's variants in terms of performance and cost. Target Architecture and Specifications are described in this chapter. Time in system nanosecond needed for creating message authentication code (MAC) using every variants of MAC generation algorithm implemented in java is calculated and performance is analyzed by calculating cycle per byte.

5.1 Target Architectures

The main goal of this thesis is to measure the performance of the all variants of MAC generation algorithm based on cipher block. These candidates algorithm are tested on desktop system. The following system is used:

- A PC with an Intel Core i5 -2450M Processor 2.50GHz having 4 GB RAM .the operating system is Windows 7 Ultimate running in 64-bit mode. The system is running the java VM 22.0 –b10, Java HotSpot(TM) 64-Bit Server 1.7.0_02 with NetBeans IDE 7.0.1.

5.2 Measuring Cost

There is some extra cost which may be added to the absolute cost for creating message authentication code using different CBC MAC algorithm and its all variants but this is equally affected to all candidates algorithm on the execution. The system time in nanosecond is taken just before the execution of code segment for generating message authentication code (MAC) or tag along with key generation time in each algorithm and the completion of the execution. The time spending for creating MAC is calculated by subtracting start time taken before execution from completion time taken after completing execution of specific code segment which is used to produce MAC[19,24]. The time required for each algorithm is calculated as follows:

```
long     startTime = System.nanoTime();
//  createMAC function call
long     timeRequired  =  System.nanoTime() -  startTime;
```

Various processes may be run in background of system so absolute measurement may not be measure due to this reason, time needed for creating MAC in all algorithm may not be observe same in every run of program. Therefore at least 5 times, the program implemented in java is run in architecture describe as above section 5.2 and finally average required time observed in every run is calculated as:

Average required Time = $\sum_{i=1}^{5} \frac{T_i}{5}$ where T_i represent time obtained in ith run of execution.

This average calculated time is used to calculate cycle per byte.

5.3 Measuring Performance

Timing cryptographic primitive is useful when analyzing the performance of multiple algorithms on a single machine but it may vary on other machine therefore ,cryptographer prefer to measure how many cycle it takes to process each byte. Different cycle/byte is calculated in the same box also because of background other process. So to optimize such extra cost, average is taken running multiple times in same machine for each candidate algorithms.

In this thesis Cycle/byte calculation with the following parameters: time in second spent creating MAC(T_s), frequency of the CPU in Hz(F) and message input in bytes(L). the formula for creating cycle/byte suggested by [21] is:

$$\text{Cycle/byte} = \frac{T_s * F}{L}$$

5.4 Analysis

This section will present the result of the performance tests for various input sizes of each algorithm. A simple multiple histogram for each candidates variants will be presented each for representing cycle per byte.

Following table and corresponding charts show the overall performance in the different encryption algorithm, Triple – DES and AES. Every candidate algorithms are taken different sizes which are 5KB, 2KB, and 1KB, 595 bytes and 29 bytes.

Candidate Algorithms	**1st run AES**	**2nd run AES**	**3rd run AES**	**4th run AES**	**5th run AES**	**Average**
CBC MAC	2560859	3472259	2398418	2288745	2542173	2652490.8
EMAC	2316526	2857215	2818852	2271616	2536192	2560080.2
XCBC	11515496	11665691	13885567	11367105	16183408	12923453
TMAC	3403142	3656888	3659436	3505525	3391003	3523199
OMAC	2797653	3303130	3291486	2716398	2431614	2908056.2

Table 5.1: Performance of CBC MAC with its variants for small message size (29 byte) using Encryption Algorithm AES

Candidate Algorithms	**1st run TDES**	**2nd run TDES**	**3rd run TDES**	**4th run TDES**	**5th run TDES**	**Average**
CBC MAC	52664600	44744767	55638602	48641495	51672227	50672338.2
EMAC	4554521	5154631	4544931	4462437	4599255	4663155
XCBC	56999951	49107758	59263746	52784183	56848057	55000739
TMAC	54389157	46445896	55949044	49364089	52602773	51750191.8
OMAC	53355555	45667669	56386535	48666445	52319617	51279164.2

Table 5.2: Performance of CBC MAC with its variants for small message size (29 byte) using Encryption Algorithm TDES.

Algorithms	Cycle/byte	
	TDES	AES
CBC MAC	50672338	2652490
EMAC	4663155	2560080
XCBC	55000739	12923453
TMAC	51750191	3523199
OMAC	51279164	2908056

Table 5.3: Performance of CBC MAC with its variants for small message size (29 byte) with encryption algorithm AES and TDES.

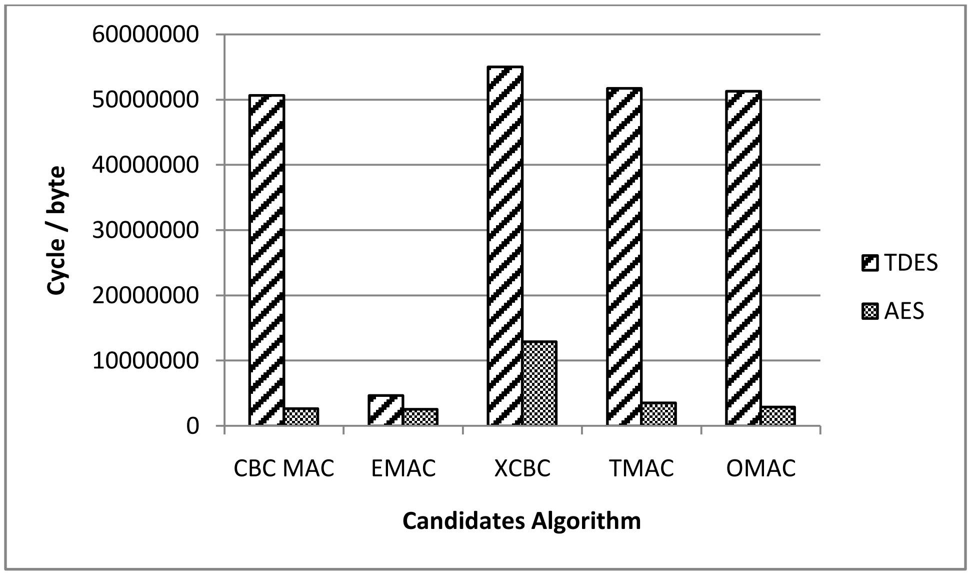


Figure 5.1: Performance of CBC MAC with its variants for small message size (29 byte) with encryption algorithm AES and TDES.

Algorithms	**1st run AES**	**2nd run AES**	**3rd run AES**	**4th run AES**	**5th run AES**	**Average**
CBC MAC	812591	747108	681382	740691	697700	735894.4
EMAC	401487	330147	798608	353485	705515	517848.4
XCBC	728772	634807	639905	577545	559669	628139.6
TMAC	307032	293452	265642	353170	258953	295649.8
OMAC	253319	197514	237403	236674	220934	229168.8

Table 5.4: Performance of CBC MAC with its variants for small message size (595 byte) using Encryption Algorithm AES

Algorithms	**1st run TDES**	**2nd run TDES**	**3rd run TDES**	**4th run TDES**	**5thrun TDES**	**Average**
CBC MAC	3609216	3609216	3609216	3609216	3609216	3609216
EMAC	1018871	1105301	962626	1090380	1213651	1078165.8
XCBC	2985831	2992532	3174044	3046302	3202320	3080205.8
TMAC	2640076	2782071	2812220	2772198	2940946	2789502.2
OMAC	2650044	2815230	2821026	2718276	2986962	2798307.6

Table 5.5: Performance of CBC MAC with its variants for small message size (595 byte) using Encryption Algorithm TDES

Algorithms	**Cycle / byte**	
	TDES	**AES**
CBC MAC	3609216	735894
EMAC	1078165	517848
XCBC	3080205	628139
TMAC	2789502	295649
OMAC	2798307	229168

Table 5.6: Performance of CBC MAC with its variants for small message size (595 byte) with encryption algorithm AES and TDES.

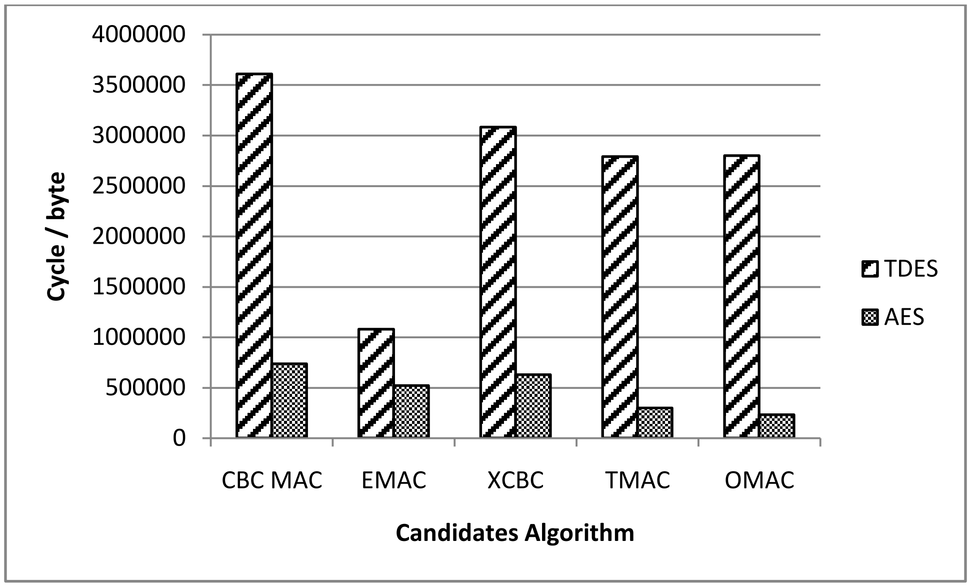


Figure 5.2: Performance of CBC MAC with its variants for small message size (595 byte) with encryption algorithm AES and TDES.

Algorithms	1st run AES	2nd run AES	3rd run AES	4th run AES	5th run AES	Average
CBC MAC	780805	951127	999286	473567	823038	805564.6
EMAC	272403	298417	259151	220089	283516	266715.2
XCBC	452112	494028	439261	330465	523558	447884.8
TMAC	205960	139272	137144	163926	174299	164120.2
OMAC	181475	213754	222420	168815	205225	198337.8

Table 5.7: Performance of CBC MAC with its variants for message size (1KB) using Encryption Algorithm AES

Algorithms	1st run TDES	2nd run TDES	3rd run TDES	4th run TDES	5th run TDES	Average
CBC MAC	2097151	2097151	2097151	1048575	2097151	1887435.8
EMAC	959655	790816	725468	389703	820509	737230.2
XCBC	1888820	1999301	1913930	1048575	1958865	1761898.2
TMAC	1711732	1957498	1788722	1048575	1886022	1678509.8
OMAC	1733230	1753478	1704440	1048575	1733683	1594681.2

Table 5.8: Performance of CBC MAC with its variants for message size (1KB) using Encryption Algorithm TDES

Algorithms	**cycle / byte**	
	TDES	**AES**
CBC MAC	1887435	805564
EMAC	737230	266715
XCBC	1761898	447884
TMAC	1678509	164120
OMAC	1594681	198337

Table 5.9: Performance of CBC MAC with its variants for message size (1KB) with encryption algorithm AES and TDES.

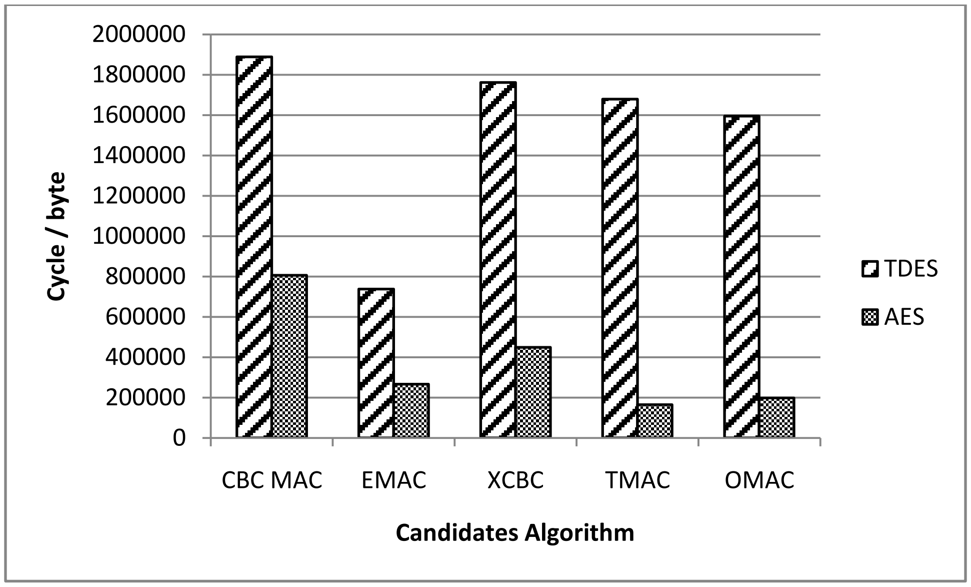


Figure 5.3: Performance of CBC MAC with its variants for message size (1KB) with encryption algorithm AES and TDES.

Algorithms	**1st run AES**	**2nd run AES**	**3rd run AES**	**4th run AES**	**5th run AES**	**Average**
CBC MAC	421614	428305	525832	473567	593367	488537
EMAC	211929	209150	232618	220089	204782	215713.6
XCBC	271567	250289	248940	330465	247749	269802
TMAC	142334	157952	150727	163926	119091	146806
OMAC	154178	150081	146286	168815	119173	147706.6

Table 5.10: Performance of CBC MAC with its variants for message size (2KB) using Encryption Algorithm AES

Algorithms	**1st run TDES**	**2nd run TDES**	**3rd run TDES**	**4th run TDES**	**5th run TDES**	**Average**
CBC MAC	1048575	1048575	1048575	1048575	1048575	1048575
EMAC	383088	341246	336103	389703	395228	369073.6
XCBC	982319	1048575	980785	1048575	1048575	1021765.8
TMAC	1002755	1005035	977885	1048575	1031203	1013090.6
OMAC	918836	975581	969754	1048575	941789	970907

Table 5.11: Performance of CBC MAC with its variants for message size (2KB) using Encryption Algorithm TDES

Algorithms	cycle/byte	
	TDES	**AES**
CBC MAC	1048575	488537
EMAC	402734	215714
XCBC	699050	269802
TMAC	693431	146806
OMAC	681573	147707

Table 5.12: Performance of CBC MAC with its variants for message size (2KB) with encryption algorithm AES and TDES.

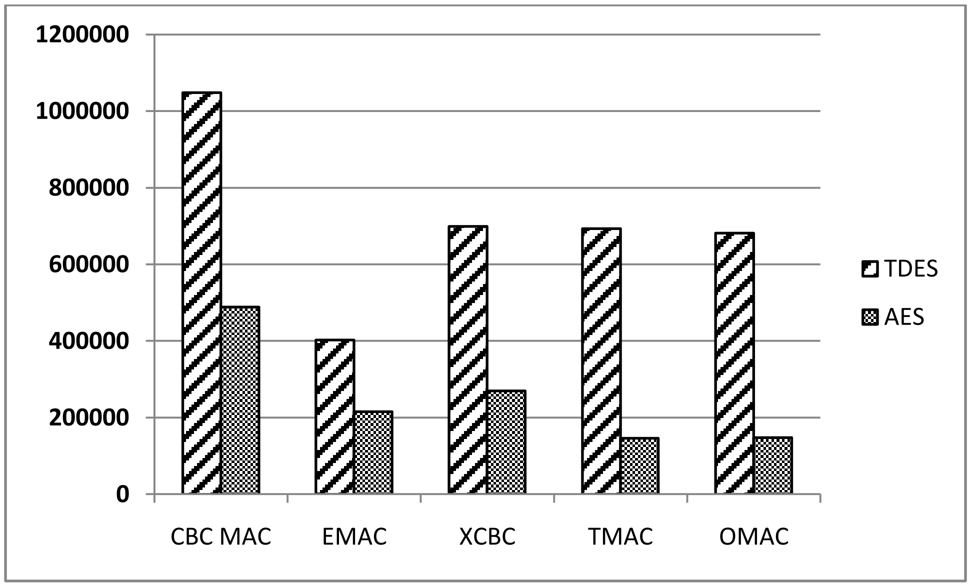


Figure 5.4: Performance of CBC MAC with its variants for message size (2KB) with encryption algorithm AES and TDES.

Algorithms	**1st run AES**	**2nd run AES**	**3rd run AES**	**4th run AES**	**5th run AES**	**Average**
CBC MAC	293302	300382	268443	321748	330845	302944
EMAC	257365	272010	233095	225892	364932	270658.8
XCBC	216471	261080	255027	214673	280137	245477.6
TMAC	125830	132991	146547	144340	155176	140976.8
OMAC	123897	146217	124667	134708	188793	143656.4

Table 5.13: Performance of CBC MAC with its variants for larger message size (5KB) using Encryption Algorithm AES

Algorithms	**1st run TDES**	**2nd run TDES**	**3rd run TDES**	**4th run TDES**	**5th run TDES**	**Average**
CBC MAC	699050	699050	699050	699050	699050	699050
EMAC	367792	378096	356210	347920	563656	402734.8
XCBC	699050	699050	699050	699050	699050	699050
TMAC	699050	699050	670958	699050	699050	693431.6
OMAC	680594	671916	668709	699050	687596	681573

Table 5.14: Performance of CBC MAC with its variants for larger message size (5KB) using Encryption Algorithm TDES

Algorithms	Cycle / byte	
	TDES	AES
CBC MAC	699050	302944
EMAC	402734	270659
XCBC	699050	245477
TMAC	693431	140976
OMAC	681573	143656

Table 5.15: Performance of CBC MAC with its variants for message size (5KB) with encryption algorithm AES and TDES.

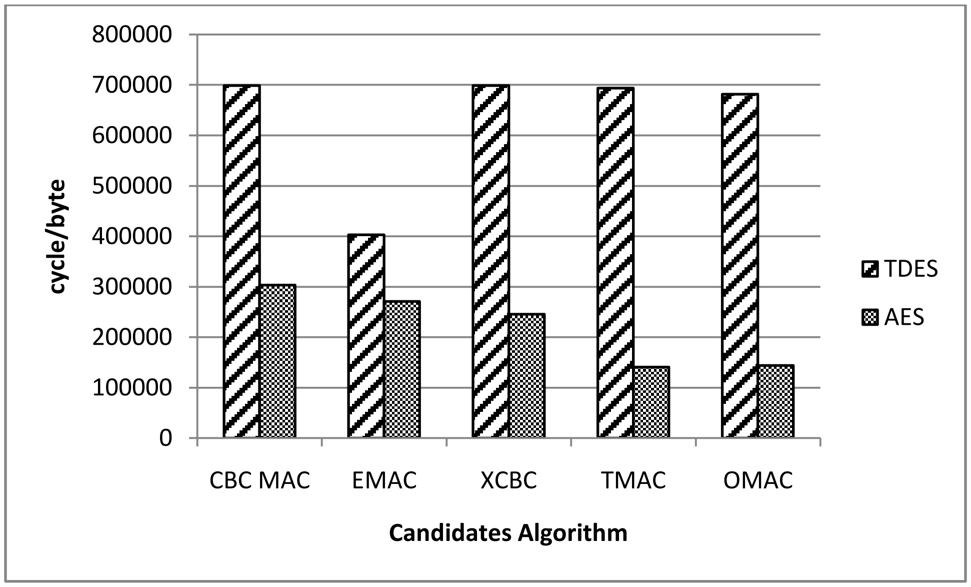


Figure 5.5: Performance of CBC MAC with its variants for small message size (5KB) with encryption algorithm AES and TDES

5.5 Result

Measuring all cipher based message authentication code algorithm in above architectural machine mansion in section 5.1 using different symmetric encryption algorithms, AES and Triple-DES, it is observed that: if the encryption algorithm is TDES, Encrypted MAC (EMAC) seems to be the better cryptographic variants of CBC MAC whatever the size of message is taken to generate message authentication code (MAC). But the nature of time required to create MAC changed according to input size of message in the case of AES. For small size of message, EMAC seems to be best model in AES algorithm also but it does not remain same when input size increased. By this way, Two-key MAC (TMAC) seems to the best cryptographic message authentication code algorithm based on cipher blocking chaining (CBC) mode. It is observed that TMAC yields 3%-80% better performance (cycle/byte) than other variants when encryption algorithm is AES for small size message when size is increased it yields to 2%-53%. EMAC yields 90%-91% better performance than other variants when encryption algorithm is TDES for small size message when size is increased it yields to 40%-42%. For comparing two symmetric encryption algorithms, TDES and AES, it seems AES algorithm is best to use for every variants of CBC MAC. TDES seems 5-11 times higher than AES.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

In this thesis, the variants of cipher block chaining based message authentication code algorithm are discussed along by using their java implementation. The result of empirical performance comparison shows that two variants of CBC MAC perform better depending on the message size and symmetric encryption algorithm. First EMAC or Encrypted MAC needs fewer cycles for processer to process byte when symmetric encryption algorithm is triple DES. In this encryption algorithm EMAC remains same performance by comparing with other variants but in case of AES result is not same for all size of message. For smaller size EMAC shows better performance than other when size of message is increased, TMAC shows the better performance. Comparatively OMAC is also shows better performance when large size of message is used to create MAC than other variants like CBC MAC, XCBC, and EMAC if the encryption algorithm is AES. For TDES for every size of input to the algorithm, EMAC shows better performance.

6.2 Future Works

The main focus of this work has been to analyze the four variants of CBC MAC and CBC MAC itself. Hence special effort could be given for analyzing the security of the candidates. After this work , it seems that all algorithms has equal cost for creating MAC by making the chain of cipher block but more cost is differ in generating extra key for security reason for variable length of message. So future work can be done to minimize the number of block cipher invocation so that it can be used in resource constrained device and environment. It can be further study to Optimize for both speed and size without reducing the security margin. A possibility here can be to create three implementation for each candidate, one for optimized for size, one for optimized for speed and one for providing better security.

References:

[1] Bellare, M., Kiliam, J., and Rogaway, P., *the Security of the cipher block chaining message authentication code*. JCSS,vol .61, no. 3,2000.

[2] Black, J., Rogaway, P., CBC MACs *for arbitrary-length messages: The three key constructions.* Advances in Cryptology — CRYPTO 2000, LNCS 1880, pp. 197–215, Springer- Verlag, 2000.

[3] Black, J., Rogaway, P., *Comments to NIST concerning AES modes of operations: A suggestion for handling arbitrary-length messages with the CBC MAC.* NIST submission. Available at http://www.cs.ucdavis.edu/~rogaway/.

[4] Coremen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., *Introduction to Algorithms*, Second Edition, Prentice-Hall, India, 2007.

[5] Dworkin, M., *Recommendation for block cipher modes of operation: the RMAC authentication mode*.NIST special publication 800-38B, Available at http://csrc.nist.gov/encryption/modes/.

[6] FIPS PUB 197, "*Advanced Encryption Standards (AES)," Federal Information Processing Standard (FIPS),*Publication AES Draft, National Institute of Standards and Technology, US Department of Commerce, Washington D.C., 2001

[7] Guerin , R., Bellare, M., and Rogaway, P., XOR MACs: *new methods for message authentication using finite pseudorandom functions Advances in cryptology – CRYPTO* ,95 ,LNCS 963 , pp, 15-28, Springer – Verlag ,1995.

[8] H.C.A.V. Tilborg, *Fundamentals of Cryptology*, Kluwer Academic Publisher Boston, 1988.

[9] Iwata, T. and Kurosawa, K., OMAC: *One-Key CBC MAC. Pre-proceedings of Fast Software Encryption,* FSE 2003, pp. 137–161, 2003.

[10] Iwata, T. and Kurosawa, K., *Stronger security bounds for OMAC, TMAC and XCBC. Manuscript Available at Cryptology ePrint Archive,* Report 2003/082

http://eprint.iacr.org/.

[11] Jueneman, R. R., Matyas, S. M., and Meyer, C. H., "*Message Authentication",* IEEE Communication, Vol 23, No. 9, pp 29-40,1985

[12] Kurosawa, K., Iwata, T., TMAC*: Two-key CBC MAC*. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 33–49. Springer, Heidelberg ,2003

[13] Minematsu and Matsushima, T., *Improved Security Bounds for PMAC, TMAC, and XCBC*. Fast Software Encryption 2007.

[14] Mitchell, C., *Partial Key Recovery Attacks on XCBC ,TMAC and OMAC , Cryptography and Coding,* 10th IMA International Conference – CCC 2005,LNCS 3796 , pp. 155-167 ,Springer –Verlag ,2005

[15] Mridul Nandi, *A Unified Method for Imporving PRF Bounds for class of block cipher based MACs (2010),* 212-219, FSE 2010, Lecture Notes in Computer Science, 6147, 2010.

[16] Netbeans ide 7.1 features , May 2010.
http://netbeans.org/features/index.html

[17] Palash Sarkar, *Pseudo-Random Functions and Parallelizable Modes of Operations of a Block Cipher* 2009. URL: http://eprint.iacr.org/2009

[18] Petrank, E., Rackoff, C., *CBC MAC for real-time data sources*. J.Cryptology, vol. 13, no. 3, pp.315–338, Springer-Verlag, 2000.

[19] Rogaway, P., Black, J., *A block- cipher mode operation for parallelizable message Authentication.* Advances in cryptology – EUROCRYPTO 2002 ,LNCS 2332, pp . 384-397, Springer – Verlag ,2002.

[20] Sun Microsystems Inc., http://java.sun.com/javame.

[21] *Timing Cryptographic Primitives,* http:// etutorials.org

[22] Venkatesan, R., Janaka Deepakumara, Howard M. Heys *Performance Comparison of Message Authentication Code (MAC) Algorithms for the Internet Protocol Security (IPSEC),*Electrical and Computer Engineering Memorial University of Newfoundland St. John's, NL, Canada, A1B3S7

[23] William Stallings *Cryptography And Network Security Principles and Practice* , Prentice all , Fifth Edition ,2010.

[24] Wikipedia, the free encyclopedia, www.wikipedia.org/