Technische Universität München
Lehrstuhl für Datenverarbeitung
Univ.-Prof. Dr.-Ing. K. Diepold

High Speed Cryptography for Network and Disk Encryption Applications

Mohamed Abo El-Fotouh

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:	**Univ.-Prof. Dr.-Ing.habil. G. Rigoll**
Prüfer der Dissertation:	**1. Univ.-Prof. Dr.-Ing. K. Diepold**
	2. Univ.-Prof. Dr.rer.nat. C. Eckert

Die Dissertation wurde am **16.06.2009** bei der Technischen Universität München eingerichtet und durch die Fakultät für Elektrotechnik und Informationstechnik am **22.10.2009** angenommen.

Preface

The research presented in this thesis has been carried out at the Institute for Data Processing (LDV) at Technische Universität München (TUM). This was during my stay as a DAAD (Deutscher Akademischer Austausch Dienst) scholarship holder. Throughout this period, I had the opportunity to meet and collaborate with several persons who have contributed to its accomplishment.

First of all, I am deeply thankful to Prof.Dr.-Ing. Klaus Diepold for his great care, valuable advices, helpful guidance, and providing the different facilities to carry out this work. He was always a frequent source of support and guidance in all aspects and at all times. I am deeply thankful to Prof. Dr. Claudia Eckert for the interest she showed to co-advise my work. I deeply appreciate DAAD who have granted me the scholarship. Furthermore I am deeply thankful to my colleagues at LDV that have provided the environment for sharing their experiences about the problems involved.

Finally I want to thank my family and my friends. The encouragement and support from them is a powerful source of inspiration and energy. A special thought is devoted to my parents for never-ending support.

Munich, Germany Mohamed Abo El-Fotouh
June 2009.

Abstract

In this thesis, new ideas in cryptography, cryptanalysis and designing high speed secure applications are presented. The Advanced Encryption Standard (AES) is used to demonstrate these ideas. In the field of cryptography, new encryption models are proposed. In the field of cryptanalysis, a secure key schedule for AES is presented together with new generalized attacks. These generalized attacks are applied on AES. The proposed encryption models are used to design and develop new network encryption schemes and new modes of operation dedicated to disk encryption applications.

Three new encryption models are proposed. All the proposed models share a main idea, which is splitting the encryption key into a primary and a secondary key. The secondary key together with the primary key are used to determine how the plaintext will be encrypted. The main functionality of the secondary key is to change the way the block cipher behaves, in other words by encrypting two identical plaintexts with the same primary key but two different secondary keys, the result is two different ciphertexts. The guidelines to use these models securely are also presented and the possibility to combine these models with each other is discussed.

In order to increase the security of the proposed encryption models, a generalized secure key schedule for block ciphers is proposed. This key schedule uses a secure cipher in the counter mode, and eliminates related-key attacks. It also increases the complexity of the exhaustive key search attack. Additionally, new ideas in cryptanalysis are proposed, which can improve some chosen plaintext, chosen ciphertext, chosen plaintext-adaptive chosen ciphertext and chosen ciphertext-adaptive chosen plaintext attacks. Using these new ideas, attacks on 5- and 6-round AES are mounted; to the best of the author's knowledge, these attacks use the least amount of chosen plaintext in the literature to attack 5- and 6-round AES.

To demonstrate the usage of the proposed encryption models in realistic scenarios, new network encryption schemes and disk encryption modes of operation are developed. These network encryption schemes allow a higher throughput than the current solutions. Furthermore, these schemes require much lower memory, which increases the number of concurrent clients a server can serve. For disk encryption applications, several new encryption modes of operation are developed and some of the current modes of operation are improved. These new modes of operation offer a higher throughput in accessing disks than current solutions.

Contents

Chapter 1

Introduction

1.1 Problem Statement

Symmetric encryption is the backbone of cryptography, and it is the most fundamental cryptographic task. It is used in a large variety of applications, like disk encryption, securing communications, encrypting e-mail messages, and many other applications [28]. Cryptographic overheads associated with symmetric encryption can be a burden for many applications. These overheads are mainly performance overheads and/or memory requirements. In some applications, trading memory resources for performance is totally acceptable; an example of these applications is disk encryption. The performance of disk encryption applications is very critical, as a noticeable slow down in the computer performance will face users' resistance to its deployment [60]. In other applications, this trade off might constrain the system scalability; e.g. for a network server: the number of concurrent clients is directly proportional to the amount of available memory on the server. To be able to achieve high throughput together with low memory consumption, new encryption models need to be developed.

1.2 Motivation

The work in this thesis is motivated with two main applications, where symmetric encryption overheads play an important role for their operation. These applications are network and disk encryption. In the following text, the motivation behind each application will be discussed separately.

1.2.1 Network Applications

The number of internet users is continuously increasing world wide. Recent statistics reported that the current number of internet users is more than 1.6 billion. This number has increased more than 350% in the last eight years and is still increasing daily [152]. Consequently, internet and network applications need to serve an increasing number of concurrent clients. The enhanced quality and performance expected from modern applications require more bandwidth capacity to meet the clients' needs. Today, modern networks have to fulfill the

demand of higher transmission rates and at the same time they have to provide data security and especially data confidentiality [84].

Key agility is particularly important in applications where only several blocks of data are encrypted between two consecutive key changes. IPSec [91, 92, 93] and ATM [41, 153, 69, 10], with small packet sizes, and consecutive packets encrypted using different keys, are two widespread protocols in which the key setup latencies may play a very important role [63].

Ciphers that require round keys pre-computation have a lower key agility due to the pre-computation time, and they also require extra RAM to hold the pre-computed round keys. This RAM requirement does not exist in the implementations of ciphers, which compute their round keys during the encryption/decryption operation (on-the-fly) [150].

There are two widely used schemes for generating the cipher's round keys in network applications. The first scheme uses round keys pre-computation and the second uses on-the-fly round keys computation. The analyses carried out in this thesis point out some shortcomings in both schemes, as the scheme that uses on-the-fly round keys computation performs more operations. On the other hand the scheme that uses round keys pre-computation uses more memory, which may limit the number of concurrent clients and is subjected to more cache misses and page faults causing degradation in system performance. Thus, new secure encryption schemes that possess both high encryption speed and low memory requirements are needed.

1.2.2 Disk Encryption Applications

In today's computing environment, the confidentiality of stored information faces a lot of threats, especially on end-user devices (e.g. personal computers, PDAs, USB sticks or external hard drives). A common threat against these devices is device loss or theft, which can lead to identity theft and other frauds. An adversary with physical access to a device may view or copy the information stored on that device. Another concern is insider attacks, such as an employee attempting to access sensitive information stored on another employee's device [128].

Usually the data stored on the PC is often significantly more valuable to a corporation than the PC itself [60]. To prevent the disclosure of sensitive data, the data needs to be secured. Disk encryption applications are usually used to protect the data on the disk by encrypting it, where all the data is encrypted with a single/multiple key(s) and encryption/decryption are done on-the-fly.

Disk encryption applications usually encrypt/decrypt a whole sector at a time. There exist dedicated block ciphers to encrypt whole sectors at a time like Bear and Lion [5]. These ciphers are considered to be slow, as they process the data through multiple passes. Another method is to let a block cipher like the Advanced Encryption Standard (AES) [121] (with 16 bytes as a block size) process the data using a particular mode of operation. There exist two main classes of disk encryption modes of operation, namely narrow-block and wide-block modes. The narrow-block modes operate on relatively small portions of data (typically 16 bytes when AES is used), while the wide-block modes encrypt or decrypt a whole sector (typically 512 bytes) at a time [134].

The current encryption modes of operation are either expensive in terms of performance, or are subjected to some attacks. Thus, new secure and fast

modes of operation are needed.

1.3 Original Contributions

In this thesis, novel encryption models are proposed. These models share a main idea, which is providing a block cipher with a secondary key. The secondary key together with the primary key (main encryption key) are used to determine how the plaintext will be encrypted. The main functionality of the secondary key is to change the way the block cipher behaves. In other words by encrypting two identical plaintexts with the same primary key and two different secondary keys, the result will be two different ciphertexts. It can be thought that the primary key together with the secondary key form the encryption key. These encryption models are used to design ciphers that accept multiple keys. The proposed ciphers are designed to be efficiently deployed in network encryption schemes that possess high speed together with low memory requirements, and to develop fast encryption modes of operation for disk encryption.

The proposed encryption models mainly change the cipher's expanded key by permutation, substitution or addition. These models possess a dynamic nature, i.e. they can use extra inputs to determine how the secondary key will affect the cipher's expanded key. Although these models are general and can be applied to any block cipher, this thesis focuses on their implementation using AES [121, 37, 38].

Using the proposed encryption models, new variants of AES are proposed. These variants allow AES to accept secondary key(s). With the help of these variants, new efficient network encryption schemes are proposed. These encryption schemes are characterized by their high throughput, stability, low setup time, low memory requirements and scalability. The new proposed variants of AES are used to design new modes of operation for disk encryption applications. These modes of operation are characterized by their high throughput and security, when compared to the state of the art modes.

New ideas in cryptanalysis are proposed, which can increase the efficiency of some chosen plaintext/ciphertext attacks. Based on these ideas, the Push-down attacks on AES are developed and the Substitution Cipher Chaining mode (SCC) [49] is broken.

An enhanced key schedule for AES is proposed. This proposed key schedule protects the current AES implementation from many attacks like related-key and some cache timing attacks. It is worth mentioning that the proposed key schedule increases the time complexity of many attacks, even the exhaustive key search attack. A generalized secure key schedule for block ciphers is also presented.

1.4 Thesis Organization

This thesis addresses different topics in computer security, like encryption models, network encryption schemes and disk encryption. Each topic has its own research field where intensive studies have been conducted. The literature review of each topic will be presented in the chapter that discusses that topic. The thesis is structured as follows:

Chapter 2 presents the literature review on encryption, modes of operation and AES.

Chapter 3 presents the literature review on the known attacks on AES, together with the new proposed attacks. A new key schedule for AES is proposed, which protects AES from related-key attacks, together with other attacks. In addition, new ideas in the chosen plaintext/ciphertext attacks are proposed, these ideas can increase the strength of some attacks; using these ideas new attacks on AES are developed.

Chapter 4 presents the proposed encryption models. There are three main models: Dynamic Substitution Model (DSM), Dynamic Injection Model (DIM) and Dynamic Permutation Model (DPM). The guidelines on how to use these models securely are also presented. In addition to these main models, the possibility of constructing hybrid models is discussed (i.e. using different combinations of these models). Different variants of AES based on these models are proposed.

Chapter 5 presents the current network encryption schemes that use Cipher Block Chaining mode of operation (CBC) [117] and Counter mode of operation (CTR) [113] to perform encryption. The authenticated encryption schemes that use Galois/Counter Mode (GCM) [116] to perform authenticated encryption are also studied. After studying the existing schemes, new schemes are proposed; these schemes enjoy high-throughput, stability and scalability. To be able to realize the behavior of all the schemes, a network simulation tool is developed to realize how these schemes perform in real world situations. A stress test is developed, which investigates the performance of these schemes, when the server is overloaded. Finally, the security analyses of the current and proposed schemes are presented.

Chapter 6 presents the state of the art disk encryption modes of operation, together with the proposed modes of operation. The evaluation of these modes of operation in terms of performance and security is presented.

Chapter 7 summarizes the thesis, reviews the main contributions, and states the future work.

Parts of this thesis have been published in [47, 44, 42, 46, 45, 49, 43, 48, 51, 56, 53, 50, 52, 54, 55, 57].

Chapter 2

The Advanced Encryption Standard (AES)

2.1 Basic Definitions

Plaintext: intelligible text or signals that have meaning.

Ciphertext: plaintext that has been encrypted.

Confusion: a term introduced by Shannon [147] to describe a function whose output is a complex function of its inputs (usually composed of a data block and a round key).

Diffusion: a term introduced by Shannon [147] to describe a linear function whose output is a permutation (or wire-crossing) of its inputs and which distributes input bits so as to remove any local clustering.

Encryption: the transformation of plaintext into ciphertext for the purpose of security or privacy.

Decryption: the transformation of ciphertext into plaintext.

Encryption algorithm: an algorithm that performs encryption, it is used interchangeably with the word "cipher".

Key: a sequence of bits which is used for the transformations between ciphertext and plaintext.

2.2 Encryption Algorithms

Encryption algorithms can be classified into two main categories: asymmetric and symmetric algorithms, depending on the technique and approach employed.

2.2.1 Asymmetric Ciphers

Also known as Public key ciphers is a group of ciphers that possess two keys, one is used for encryption and the other is used for decryption. In public key

ciphers, a user has a pair of cryptographic keys, a public key and a private key. The private key is kept secret, while the public key may be widely distributed. Messages encrypted with the recipient's public key can only be decrypted with his corresponding private key. The keys are related mathematically, but the private key cannot be practically derived from the public key. Well known asymmetric ciphers are RSA [139], Diffie-Hellman [40], ElGamal [64], DSA [2], and Elliptic curve ciphers [97, 118].

Asymmetric encryption is relatively slow and therefore unsuitable for encryption of large messages. However, a major advantage of asymmetric key systems is that one of the two components of the key pair can be made public (hence the phrase "public key"). This has two important benefits: first, anyone can send private information to a recipient 'A' by encrypting the information using A's public key but only A will be able to recover the information by decrypting the ciphertext using the related private key (which A must keep secret). Second, if some known information can be correctly recovered by decrypting it with A's public key, it must have been encrypted with A's private key and therefore by A. This means that asymmetric algorithms provide proof of origin.

Figure 2.1 shows the asymmetric key encryption and decryption processes, where anyone can encrypt the data using the public key, but only the holder of the private key can decrypt that data. Security depends on the secrecy of the private key.

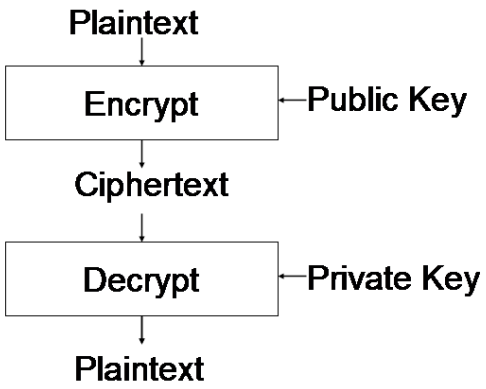


Figure 2.1: Asymmetric encryption and decryption processes.

2.2.2 Symmetric Ciphers

Also known as Secret-key ciphers, these ciphers use the same key for encryption and decryption. They usually operate at relatively high speed and are suitable for bulk encryption of messages. Symmetric key algorithms can be divided into stream ciphers and block ciphers. Stream ciphers usually encrypt the bits of the message one at a time, whereas block ciphers take a number of bits and encrypt

them as a single unit. The Advanced Encryption Standard algorithm approved by NIST in December 2001 uses 128-bit blocks [120].

Some examples of popular block ciphers include AES (Rijndael) [38], Blowfish [144], Tirple DES [33], IDEA [100], RC5 [138], CAST [3] and Skipjack [15]. Some examples of popular stream ciphers are RC4 [119], PANAMA [35] and Rabbit [24].

Figure 2.2 shows the symmetric key encryption and decryption processes, where only the persons sharing the secret key can encrypt and decrypt the message. Security depends on the secrecy of the secret key.

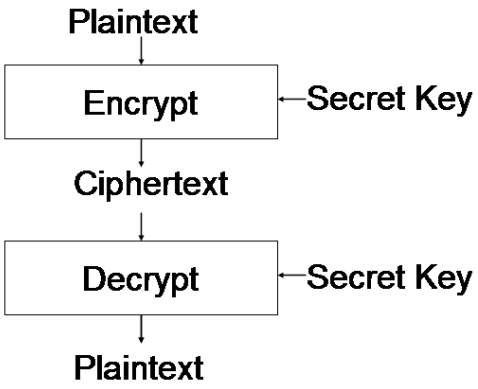


Figure 2.2: Symmetric encryption and decryption processes.

2.3 The Advanced Encryption Standard Process

Advanced Encryption Standard (AES), the block cipher approved as a standard by National Institute of Standards and Technology of the United States (NIST) [131], was chosen using a process markedly more open and transparent than its predecessor, the Data Encryption Standard (DES) [59]. A new standard was needed primarily because DES has a relatively small 56-bit key which was becoming vulnerable to brute force attacks. In addition DES was designed primarily for hardware and is relatively slow when implemented in software. While Triple-DES [33] avoids the problem of a small key size, it is very slow in software, it is unsuitable for limited-resource platforms, and may be affected by potential security issues connected with the small block size of 64-bit.

NIST has been working with industry and the cryptographic community to develop an Advanced Encryption Standard (AES) to replace the Data Encryption Standard (DES). The overall goal was to develop a Federal Information Processing Standard (FIPS) that specifies an encryption algorithm capable of protecting sensitive government information. The algorithm is expected to be used by the U.S. Government and, on a voluntary basis, by the private sector [126].

On January 2, 1997, NIST announced the initiation of AES development effort and made a formal call for algorithms on September 12, 1997. The call stipulated that AES would specify an unclassified, publicly disclosed encryption algorithm(s), available royalty-free, worldwide. In addition, the algorithm(s) must implement symmetric key cryptography as a block cipher and support block sizes of 128-bit and key sizes of 128-, 192-, and 256-bit.

On August 20, 1998, NIST announced a group of fifteen AES candidate algorithms at the First AES Candidate Conference (AES1). These algorithms are CAST-256 [4], CRYPTON [104], DEAL [106, 89], DFC [137, 96], E2 [111], FROG [157], HPC [146], LOKI97 [27], MAGENTA [14, 81], SAFER+ [109], MARS [28], RC6 [99], Rijndael [37, 38], Serpent [6], and Twofish [145]. These algorithms had been submitted by members of the cryptographic community from around the world. At that conference and in a simultaneously published Federal Register notice, NIST solicited public comments on the candidates.

A Second AES Candidate Conference (AES2) was held in March 1999 to discuss the results of the analysis conducted by the global cryptographic community on the candidate algorithms. The public comment period on the initial review of the algorithms closed on April 15, 1999. Using the analyses and comments received, NIST selected five algorithms from the fifteen. AES finalist candidate algorithms were MARS, RC6, Rijndael, Serpent, and Twofish, and NIST developed a Round 1 Report [124] describing the selection of the finalists.

These finalist algorithms received further analysis during a second, more in-depth review period prior to the selection of the final algorithm(s) for AES. Until May 15, 2000, NIST solicited public comments on the remaining algorithms. Comments and analysis were actively sought by NIST on any aspect of the candidate algorithms, including, - but not limited to, - the following topics: cryptanalysis, intellectual property, crosscutting analyses of all AES finalists, overall recommendations and implementation issues. An informal AES discussion forum was also provided by NIST for interested parties to discuss AES finalists and relevant AES issues.

Near the end of Round 2, NIST sponsored the Third AES Candidate Conference (AES3) - an open, public forum for discussion of the analyses of AES finalists. AES3 was held April 13-14, 2000 in New York. Submitters of AES finalists were invited to attend and engage in discussions regarding comments on their algorithms. The final agenda for AES3 includes links to the accepted papers, their presentations, the submitter statements, and the submitter presentations. All papers proposed for AES3 were considered as official Round 2 public comments [125].

After the close of the Round 2 public analysis period on May 15, 2000, NIST studied all available information in order to make a selection for AES. On October 2, 2000, NIST announced that it has selected Rijndael to propose for AES. AES is published by NIST as FIPS PUB 197 in November 2001 [120]. In the next section, Rijndael (AES) is discussed in details.

2.4 AES Algorithm

In this thesis the names AES and Rijndael will be used interchangeably. The description in this section is mainly from [37, 38]. It is worth mentioning that the design of Rijndael was strongly influenced by the design of the block cipher

Square [36].

2.4.1 AES Encryption

AES accepts 128-bit long plaintext. The plaintext is arranged as an array of bytes. This array is called the state; the bytes of the state are treated as elements of the finite field GF(2^8). Figure 2.3 represents the state S as a matrix. AES has N rounds, where N equals to 10 for AES-128, 12 for AES-192, and 14 for AES-256. Table 2.1 presents a high-level description of AES encryption, where *State* represents the input/output state and *ExpandedKey* represents AES' expanded key.

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

Figure 2.3: AES state arranged as a matrix.

Table 2.1: AES Encryption.

```
Encrypt-AES(State,ExpandedKey)
    AK(State,ExpandedKey)
    for i=1 to N-1
        SB(State)
        SR(State)
        MC(State)
        AK(State,ExpandedKey + 4 × i)
    end for
    SB(State)
    SR(State)
    AK(State,ExpandedKey + 4 × N)
return State
```

An AES round is composed of the following four operations:

SubBytes (SB): is a non-linear byte substitution, operating on each of the *State* bytes independently. This operation provides the non-linearity in the cipher. The S-box used is derived from the multiplicative inverse over GF(2^8), known to have excellent differential and linear properties [132]. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation. The S-box is also chosen to avoid any fixed points, and also any opposite fixed points.

The inverse of SubBytes uses the inverse table. This is obtained by the inverse of the affine mapping followed by taking the multiplicative inverse in GF(2^8).

Figure 2.4 demonstrates the SubBytes transformation, where each byte in the state is replaced with its entry in a fixed 8-bit S-BOX (**S**); $b_{ij} = S(a_{ij})$.

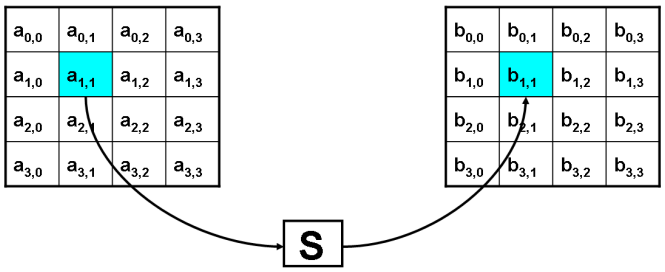


Figure 2.4: The SubBytes transformation.

ShiftRows (SR): in ShiftRows, the rows of the State are cyclically shifted over different offsets. Row 0 is not shifted, row 1 is shifted over 1 byte, row 2 is shifted over 2 bytes and row 3 is shifted over 3 bytes.

Figure 2.5 demonstrates the Shiftrows transformation, where bytes in each row of the state are shifted cyclically to the left.

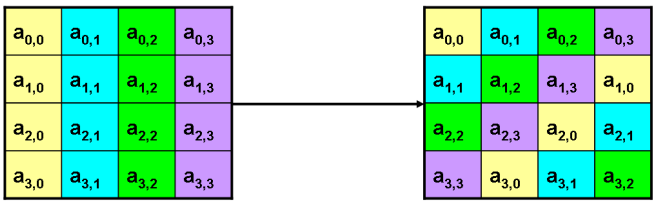


Figure 2.5: The ShiftRows transformation.

MixColumns (MC): in MixColumns, the columns of the State are considered as polynomials over GF(2^8) and multiplied modulo $x^4 + 1$ with a fixed polynomial c(x), given by:

$$c(x) = 3x^3 + x^2 + x + 2 \tag{2.1}$$

This polynomial is coprime to $x^4 + 1$ and therefore invertible. The inverse of MixColumns is similar to MixColumns. Every column is transformed by multiplying it with a specific multiplication polynomial d(x), defined by:

$$d(x) = 11x^3 + 13x^2 + 9x + 14 \tag{2.2}$$

Figure 2.6 demonstrates the MixColumns transformation, where each column of the state is multiplied with a fixed polynomial c(x).

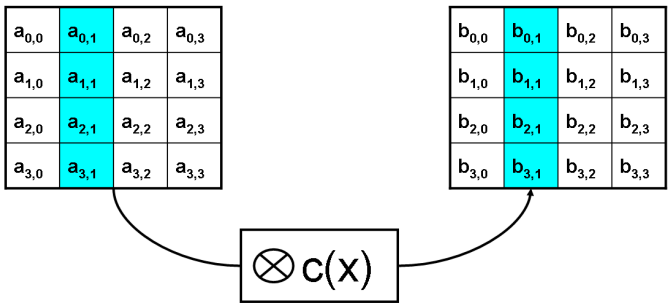


Figure 2.6: The MixColumns transformation.

AddRoundKey (AK): in AddRoundKey, a round key is applied to the State by a simple bitwise XOR. The Round Key is derived from the Encryption Key by means of the key schedule.

Figure 2.7 demonstrates the AddRoundKey operation, where each byte of the state is combined with a byte of the round key using the XOR operation ($\oplus$).

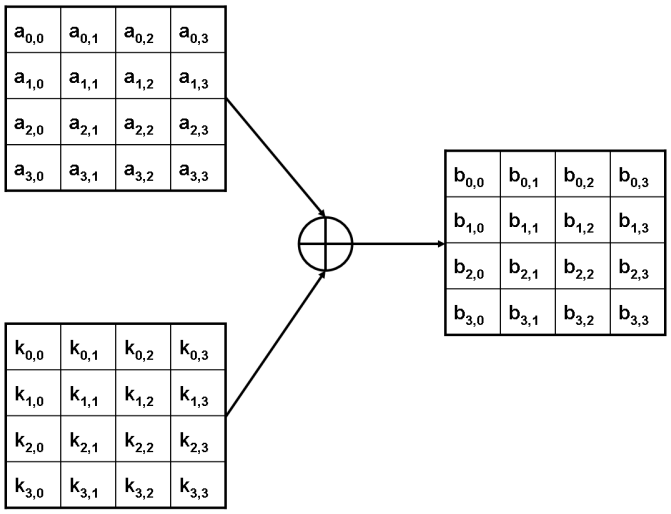


Figure 2.7: The AddRoundKey operation.

The MixColumns operation is omitted in the last round and an initial key addition is performed before the first round for whitening.

2.4.2 AES Decryption

There are two different algorithms for AES decryption routine, the inverse cipher and the equivalent inverse cipher.

The Inverse Cipher

To inverse a round, the order of the transformations in the round is reversed, and consequently the non-linear step will end up being the last step of the inverse round and the rows are shifted after the application of (the inverse of) MixColumns. Table 2.2 presents a high-level description of AES decryption, where SB^{-1}, SR^{-1} and MC^{-1} are the inverse of SB, SR and MC respectively.

Table 2.2: AES Decryption.

```
Decrypt-AES(State,ExpandedKey)
    AK(State,ExpandedKey + 4 × N)
    SR^-1(State)
    SB^-1(State)
    for i=N-1 to 1
        AK(State,ExpandedKey + 4 × i)
        MC^-1(State)
        SR^-1(State)
        SB^-1(State)
    end for
    AK(State,ExpandedKey)
return State
```

The Equivalent Inverse Cipher

Using some algebraic properties, the sequence of transformations of AES' inverse can be equal to that of AES itself, with the transformations replaced by their inverses and a change in the key schedule. This is illustrated in table 2.3, where *IExpandedKey* is a modified AES' expanded key.

Table 2.3: The Equivalent Inverse Cipher.

```
Encrypt-AES(State,IExpandedKey)
    AK(State,IExpandedKey + 4 × N)
    for i=1 to N-1
        SB^-1(State)
        SR^-1(State)
        MC^-1(State)
        AK(State,IExpandedKey + 4 × i)
    end for
    SB^-1(State)
    SR^-1(State)
    AK(State,IExpandedKey)
return State
```

Note that the key expansion for the Equivalent Inverse Cipher is defined as follows:

1. Apply the Key Expansion.
2. Apply MC^{-1} to all round keys except the first and the last one.

2.4.3 AES Key Schedule

The round keys are derived from the Encryption Key by means of the key schedule. This consists of two components: the key expansion and the round key selection. The basic principle is the following:

1. The total number of round key bits is equal to the 128-bit multiplied by the number of rounds plus 1 (e.g., for a block length of 128-bit and 10 rounds, 1408 round key bits are needed).
2. The Encryption Key is expanded into an Expanded Key.
3. Round keys are taken from this Expanded Key in the following way: the first Round Key consists of the first 16 bytes, the second one of the following 16 bytes, and so on.

For a detailed explanation of the key schedule, refer to [38].

2.5 Modes of Operations

A block cipher operates on blocks of fixed length, often 128-bit. Because messages may be of any length, several modes of operation have been invented which allow block ciphers to provide confidentiality for messages of arbitrary length.

The earliest modes described in the literature provide only confidentiality or message integrity, but do not perform both simultaneously. Examples of these modes are Cipher Block Chaining (CBC), Cipher Feedback (CFB) and Output Feedback (OFB) [117]. Other modes have been designed to offer both confidentiality and message integrity in one pass, such as IAPM [77], CCM [158], EAX [9], GCM [116], and OCB [142] modes.

Tweakable modes of operation have been developed after the introduction of tweakable block cipher [105].

Chapter 3

Cryptanalysis of AES

3.1 Attacks on the Cipher

Table 3.1 summarizes the complexities of some known attacks on AES, where CP refers to Chosen Plaintext, ACPC refers to Adaptive Chosen Plaintext and Ciphertext. The time complexity is measured in encryption units. In the following text, a brief discussion on these attacks is presented.

3.1.1 Exhaustive Key Search Attack

Exhaustive key search is the basic technique of trying all key values one by one until the correct key is found. To identify the correct key it is sufficient to know a small amount of plaintexts and their corresponding ciphertexts. If the plaintext has some known form of redundancy, such as an ASCII coded text, a small amount of ciphertext is sufficient. Exhaustive key search is an attack that does not exploit the internal structure of a cipher. Later, the attacks that exploit structural properties of the block cipher will be discussed. These types of attack are denoted by the term *cryptanalysis*. A cryptanalytic attack breaks a cipher academically if its expected workload is below that of exhaustive key search. Such an attack is called a *shortcut attack*. The existence of a shortcut attack for a given cipher does not necessarily mean that the cipher has no longer any security to offer, because most shortcut attacks described in cryptographic literature cannot be implemented in a practical setting [133].

For many modern ciphers, no shortcut attacks are known. Still, the resistance of iterative block ciphers with respect to a specific cryptanalytic method can be evaluated by performing it on reduced-round versions of the block cipher. Attacks on reduced-round versions allow getting an idea of the security margin of a cipher. If for a cipher with R rounds there exists a shortcut attack against a reduced-round version with R-r rounds, the cipher has an absolute security margin of r rounds or a relative security margin of r/R. Note that the discovery of an attack on a reduced-round version with R/2 rounds does not mean that the cipher is half-broken. Indeed, the complexity of most academic attacks increases exponentially with the number of rounds.

Table 3.1: The complexity of some attacks on AES.

Key Size	Source	Number of	Data Complexity	Time Complexity	Attack Type
all	Ref. [37]	4	2^9 CP	2^9	Square
all	Ref. [37]	5	2^{11} CP	2^{40}	Square
all	Ref. [37]	6	2^{32} CP	2^{72}	Square
all	Ref. [37]	6	2^{32} CP	2^{72}	Square
all	Ref. [61]	6	6 x 2^{32} CP	2^{44}	Partial Sums
all	Ref. [61]	7	2^{128}-2^{119}CP	2^{120}	Partial Sums
all	Ref. [159]	6	$2^{114.5}$ CP	2^{50}	Imp.Diff
all	Ref. [159]	6	$2^{75.5}$ CP	2^{104}	Imp.Diff
all	Ref. [159]	7	$2^{115.5}$ CP	2^{119}	Imp.Diff
all	Ref. [65]	7	2^{32}CP	$\approx 2^{128}$	Collision
all	This Thesis	5	2^9 CP	2^{40}	Pushdown-Square
all	This Thesis	6	2^{11} CP	2^{72}	Pushdown-Square
128	Ref. [19]	5	$2^{29.5}$ CP	2^{31}	Imp.Diff
128	Ref. [31]	6	2^{86} CP	2^{125}	Imp.Diff
128	Ref. [31]	6	$2^{91.5}$ CP	2^{122}	Imp.Diff
128	Ref. [61]	7	$2^{115.5}$ CP	2^{120}	Partial Sums
128	Ref. [21]	5	2^{39} ACPC	2^{39}	Boomerang
128	Ref. [21]	6	2^{71} ACPC	2^{71}	Boomerang
192	Ref. [136]	7	2^{92} CP	2^{186}	Imp.Diff
192	Ref. [31]	7	2^{92} CP	2^{162}	Imp.Diff
192	Ref. [61]	8	2^{128}-2^{119}CP	2^{188}	Partial Sums
192	Ref. [61]	7	19 x 2^{32} CP	2^{155}	Partial Sums
192	Ref. [107]	7	2^{32} CP	2^{184}	Square
256	Ref. [136]	7	$2^{92.5}$ CP	$2^{247.5}$	Imp.Diff
256	Ref. [31]	8	$2^{116.5}$ CP	2^{162}	Imp.Diff
256	Ref. [61]	8	2^{128}-2^{119}CP	2^{204}	Partial Sums
256	Ref. [61]	7	21 x 2^{32} CP	2^{172}	Partial Sums
256	Ref. [107]	7	2^{32} CP	2^{200}	Square

3.1.2 Differential and Linear Cryptanalysis

Linear cryptanalysis [110] attacks are possible if there are predictable input-output correlations over all but a few (typically 2 or 3) rounds significantly larger than $2^{n/2}$, where n is the block length in bits. An input-output correlation is composed of linear trails, where its correlation is the sum of the correlation coefficients of all linear trails that have the specified initial and final selection patterns. The correlation coefficients of the linear trails are signed and their sign depends on the value of the round keys. To be resistant against linear cryptanalysis, it is a necessary condition that there are no linear trails with a correlation coefficient higher than $2^{n/2}$ [37].

Differential cryptanalysis attacks [20] are possible if there are predictable difference propagations over all but a few (typically 2 or 3) rounds that have a *prop ratio* (the relative amount of all input pairs that for the given input difference gives rise to the output difference) significantly larger than 2^{1-n} if n is the block length. Difference propagation is composed of differential trails, where its prop ratio is the sum of the prop ratios of all differential trails that have the specified initial and final difference patterns. To be resistant against differential cryptanalysis, it is therefore a necessary condition that there are no differential trails with a predicted prop ratio higher than 2^{1-n} [37].

Differential and linear cryptanalyses are the two most powerful general purpose cryptographic attacks known to date. AES is based on the substitution-permutation network (SPN) structure, several research have dealt with provable security against differential and linear cryptanalyses for block ciphers based on the substitution-permutation network (SPN) structure (including AES) [32, 79, 85, 87, 86, 135, 143]. The current standard approach to demonstrate provable security against differential and linear cryptanalysis is to prove that the maximum expected differential probability (MEDP) and the maximum expected linear probability (MELP) are sufficiently small over T core rounds [132]. This is because the corresponding estimate of the data complexity of the attack (the number of plaintext-ciphertext pairs required) is proportional to the inverse of the MEDP for differential cryptanalysis and that of MELP for linear cryptanalysis.

There is a well-known duality between differential and linear cryptanalyses that often allows results for one attack to be translated into corresponding results for the other [13]. In [88], the exact MEDP of two AES rounds was determined to be $53/2^{34} \approx 1.656 \times 2^{-29}$ and that of MELP is $109,953,193/2^{54} \approx 1.638 \times 2^{-28}$. A particularly useful relationship exists for AES and related SPNs: if μ is an upper bound on the two-round MEDP (or MELP), then μ^4 is an upper bound on the MEDP (MELP) for T $\geq$ 4 [143]. This immediately yields to new upper bounds on AES MEDP and MELP for four or more rounds, namely $(53/2^{34})^4 \approx 1.881 \times 2^{-114}$ and $(109,953,193/2^{54})^4 \approx 1.802 \times 2^{-110}$ respectively.

3.1.3 Impossible Differentials

Impossible differential cryptanalysis [15] is a form of differential cryptanalysis for block ciphers. While ordinary differential cryptanalysis tracks differences that propagate through the cipher with greater than expected probability, impossible differential cryptanalysis exploits differences that are impossible (having probability 0) at some intermediate state of the cipher. AES has the follow-

ing impossible differential property: given plaintext pair which are equal at all bytes but one, the ciphertexts after 4-round cannot be equal in any of the following prohibited combinations of bytes: (1,6,11,16), (2,7,12,13), (3,8,9,14), nor (4,5,10,15).

This property follows from the property of MixColumns transformation: if two inputs of this transformation differ by one byte then the corresponding outputs differ by all the four bytes.

This property was used to attack 5-round AES-128 [19]. Later in [31], the same 4-round impossible differentials are used to attack 6-round AES-128. In [136], using the same impossible differentials, attacks on 7-round AES-192 and AES-256 are presented. In [159], the attack has been extended to 7-round AES-128 and 8-round AES-256.

3.1.4 Boomerang Attack

Boomerang attack [156] is a chosen plaintext-adaptive chosen ciphertext attack. It is an extension of differential cryptanalysis and works on quartets of data (P,P'), (Q,Q'). The attack works when encryption function E() can be split into $E = E1.E0$, where E0 is weak in encryption direction and E1 is weak in decryption direction. The boomerang attack was developed in 1999 after AES competition was already running. This attack sometimes allows breaking more rounds than the conventional differential or linear attacks, especially for the ciphers with few but carefully designed rounds. Boomerang Attack can break 5- and 6-round of AES-128.

3.1.5 Collision Attack

In [65], a collision attack was induced against AES. The existence of collisions between some partial byte oriented functions induced by AES provides a distinguisher between 4 inner rounds of AES and a random permutation, which in turn enables to mount attacks on 7-round AES for any key-length. The attack exploits (using the birthday paradox) a new kind of cryptanalytic bottleneck, namely the fact that a partial function induced by the cipher is entirely determined by a remarkably small number of unknown constants. Therefore, unlike most statistical attacks, it requires a rather limited number of plaintexts (about 2^{32}). But on the other hand, this attack requires a lot of computations.

3.1.6 Square Attack

The Square attack is a dedicated attack that exploits the byte-oriented structure of Square cipher and was published in the paper presenting the Square cipher itself [36]. This attack is also valid for Rijndael (AES) [38], as Rijndael inherits many properties from Square. The attack is a chosen plaintext attack and is independent of the specific choices of SB, the multiplication polynomial of MC and the key schedule. It is faster than an exhaustive key search for Rijndael versions of up to 6 rounds. In the following text, the square attack on 4-, 5- and 6-round AES will be presented.

Λ-Set

Let a Λ-set be a set of 256 AES states that are all different in some of the state bytes (the active) and all equal in the other state bytes (the passive), recall that the state of Rijndael is a 4×4 byte matrix. In other words for two distinct states A and B in a Λ-set:
$A_{i,j} \neq B_{i,j}$ if the byte at position (i,j) is active, and
$A_{i,j} = B_{i,j}$ else, i.e. the byte at position (i,j) is passive
A Λ-set with exactly k active bytes is a "Λ^k-set".

Square-4

Square-4 is the basic attack that attacks 4-round of Rijndael. The adversary chooses one Λ^1-set P_0 of plaintexts (where by P_i is the set of 256 states which are the output of the i^{th} round). From the round properties of Rijndael: P_1 is a Λ^4-set, P_2 is a Λ^{16}-set, and P_3 is unlikely to be a Λ-set. As explained in [38] all the bytes of P_3 are balanced, i.e. the following property holds:

$$For\ all\ (i,j) \in \{0,1,2,3\}^2 : \bigoplus_{A \in P_3} A_{i,j} = 0. \tag{3.1}$$

Hence, all bytes at the input of the 4^{th} round are balanced. This balance is in general destroyed by the subsequent application of SB. It is assumed that the 4^{th} round is a final round, i.e., it does not include a MC operation. Every output byte of the 4^{th} round depends on only one input byte of the 4^{th} round. Let a be the output of the 4^{th} round, b its input, S the S-box and k the round key of the 4^{th} round. This means that:

$$a_{i,j} = S(b_{i',j'}) \oplus k_{i,j} \tag{3.2}$$

By assuming a value for $k_{i,j}$, the value of $b_{i',j'}$ for all elements of the Λ-set can be calculated from the ciphertexts. If the values of this byte are not balanced over Λ, the assumed value for the key byte was wrong. This is expected to eliminate all but approximately 1 key value. This can be repeated for the other bytes of k. Since by checking a single Λ^1-set only 2^{-8} of the wrong key assumptions are left as possible candidates, the Encryption Key can be found with overwhelming probability with only 2 Λ-sets. The cost of this attack is 2^9 chosen plaintext and requires about 2^9 cipher executions.

Square-5

If an additional round is added at the end, the value of $b_{i',j'}$ is calculated from the output of the 5^{th} round instead of the 4^{th} round. This can be done by additionally assuming a value for a set of 4 bytes of the 5^{th} round key. As in the case of the 4-round attack, wrong key assumptions are eliminated by verifying that $b_{i',j'}$ is not balanced. In this 5-round attack 2^{40} key values must be checked, and this must be repeated 4 times. Since by checking a single Λ-set only 2^{-8} of the wrong key assumptions are left as possible candidates, the Encryption Key can be found with overwhelming probability with only 5 Λ-sets. The cost of this attack is 5×2^8 chosen plaintext and requires about 2^{40} cipher executions.

Square-6

The basic idea is to choose a set of plaintexts that results in a Λ^1-set at the output of the 1^{st} round. This requires the assumption of values of four bytes of the round key that is applied before the first round. If the intermediate state after MC of the 1^{st} round has only a single active byte, this is also the case for the input of the 2^{nd} round. A set of 2^{32} plaintexts are to be considered, such that one column of bytes at the input of MC of the first round range over all possible values and all other bytes are constant. Now, an assumption is made for the value of the 4 bytes of the relevant bytes of the first round key. From the set of 2^{32} available plaintexts, a set of 256 plaintexts can be selected that results in a Λ^1-set at the input of round 2. Now Square-5 attack can be performed. For the given key assumption, the attack can be repeated for several plaintext sets. If the byte values of the last round key are not consistent, the initial assumption must have been wrong. A correct assumption for the 4 bytes of the first round key will result in the swift and consistent recuperation of the last round key. The cost of this attack is 2^{32} chosen plaintext and about 2^{72} cipher executions.

Square attack for L-representation

The implementation of AES has a great degree of freedom to change the order of its elementary operations, without changing the behavior of the cipher. The L-representation is defined in [107], where L^r is defined as:

$$L^r = MC^{-1}(SB^{-1}(K^r)) \tag{3.3}$$

Note that by knowing L^r is equivalent to knowing K^r, where K^r is the r^{th} round key. In table 3.2, the original round of AES and its equivalent using the L-representation are presented.

Table 3.2: AES original and equivalent round functions.

Round function	Equivalent function
SB(State)	SB(State)
SR(State)	AK(State,L^r)
MC(State)	SR(State)
AK(State,K^r)	MC(State)
return State	return State

The square attack in [38] assumes that the last round is a final round (i.e. there is no MC operation). By using the L-representation as in [107], the square attack can be restated to work on full rounds. For four round AES, the attack works as following: the adversary chooses one Λ^1-set P_0 of plaintexts (where by P_i is the set of 256 states which are the output of the i^{th} round). As explained in [38] all the bytes of P_3 are balanced, i.e. the balance property holds (3.1). P_4 is the set of 256 ciphertexts the adversary learns. Let L^4 be the L-representation of K^4 (3.3). The adversary can calculate the set Q_4 in between P_3 and P_4 by applying MC^{-1} and SR^{-1} on the elements of P_4, where for any $X_i \in \mathrm{P}_4$ the corresponding element is $Z_i \in \mathrm{Q}_4$. The fourth round is inverted step by step, by inverting the MC operation , inverting the SR operation, add (a possible choice for) the key $\mathrm{L}^4_{i,j}$ and invert the SR operation. If the guess $a \in \{0,1\}^8$ for $\mathrm{L}^4_{i,j}$

is correct, the set of bytes $\mathrm{SB}^{-1}(\mathrm{Z}_{i,j} \oplus a)$ is balanced, it is estimated that this guess will eliminate all the wrong guesses but one. So, an expected number of about 2^{16} candidates of L^4 can be constructed.

Each candidate corresponds with a unique choice of the key of the last round. Another set of Λ^1-set can be tried to eliminate the wrong candidates, or just use exhaustive key search over all the key candidates using the same 256 known pairs of plaintext and ciphertext as before. With overwhelming probability, either approaches uniquely determined the last round key. The memory requirement of this attack is low and needs either 2^9 chosen plaintext and 2^9 cipher executions or 2^8 chosen plaintext and 2^{16} cipher executions.

3.1.7 Partial Sums

Partial Sums attack [61] is an extension to the Square attack. It is much faster than the Square attack to attack 6-round AES. It can also attack 7-round AES-128 and 8-round AES-192 or AES-256. But the latter attacks require nearly the entire Rijndael codebook (2^{128}-2^{119} chosen plaintexts); they are therefore not very practical even for an adversary with sufficient computing power. In the following text, the attack on 6-round AES is presented.

PartialSums-6

The attack of Sect. 3.1.6 on 6-round of AES can be improved. Instead of guessing four bytes of the first round key, all 2^{32} plaintexts are used. For any value of the first round key, these encryptions consist of 2^{24} groups of 256 encryptions that vary only in a single byte after MC of round 1. The five key bytes at the end of the cipher have to be guessed, a partial decrypt to a single byte of the state after the key addition of round 4 has to be done, this value over all the 2^{32} encryptions have be summed, and the result is checked for a zero result. Compared to the original version, only 40 bits are guessed instead of 72. On the other hand, 2^{24} more work for each guess are done. This attack requires 6×2^{32} chosen plaintexts and 2^{44} cipher executions.

3.2 Proposed Attacks

In this section, new concepts in cryptanalysis are introduced; these concepts can increase the strength of some attacks.

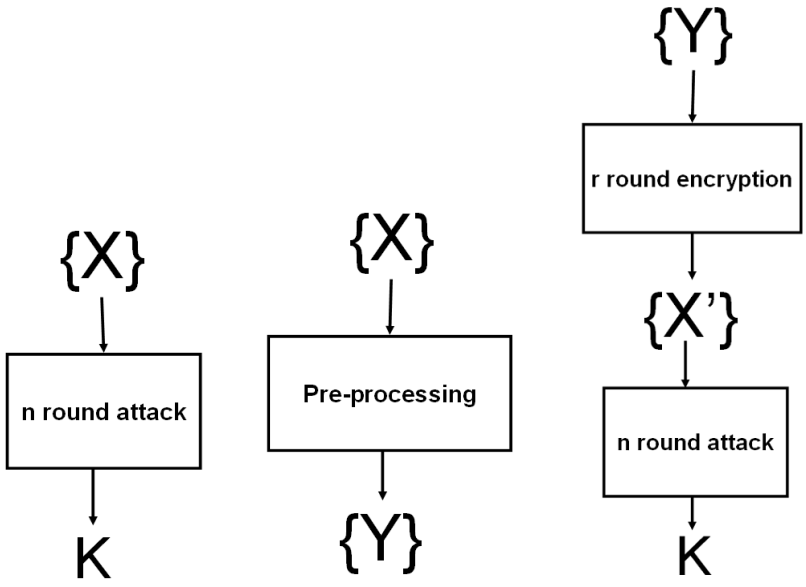
3.2.1 Pushdown Attack

The Pushdown attack hosts a chosen plaintext attack, where it prepares the chosen plaintext for that attack. Figure 3.1 gives an overview on how it works, where:

1. In Figure 3.1(a), an n round chosen plaintext attack is presented that accepts the set $\mathbf{\{X\}}$ of chosen plaintexts to calculate the Key $\mathbf{K}$.

2. Figure 3.1(b) presents the pre-processing step of the Pushdown attack, where $\mathbf{\{X\}}$ is transformed to $\mathbf{\{Y\}}$.

3. Figure 3.1(c) presents the Pushdown attack, where after encrypting **{Y}** with r rounds, the result is **{X'}** that should be equivalent to **{X}**, in the sense that applying the n round chosen plaintext attack on **{X'}** will recover **K**. Thus, the number of rounds attacked by the chosen plaintext attack is increased to **(n+r)** rounds.



(a) n round attack (b) Pre-processing (c) Pushdown attack

Figure 3.1: Overview on the Pushdown attack.

In the following text, some pre-processing steps, which are needed to mount the Pushdown attacks on AES, are presented.

Λ^1-Set Pre-Processing

Proposition 3.1. *Let* Ω^1 *be defined by:*

$$\Omega^1 = R^{-1}(\Lambda^1, 0) \tag{3.4}$$

where $R^{-1}(X, K)$ *applies an AES decryption round on the state* X *and Round key* $\boldsymbol{K}$*. Then* Ω^1*-Set is a* Λ^4*-Set.*

Proof. The proof is straight forward. Following the difference propagation, applying AK will not change the difference and the result is still a Λ^1-Set. After applying MC^{-1} the difference will propagate to one column, thus the result is a Λ^4-Set. Applying SR^{-1} will not change the difference but will change the positions of the active bytes. Finally, SB^{-1} will not change the difference, thus the result is still a Λ^4-Set. □

Note, a key with all zeros is used as a kind of optimization and simplification of the analysis, in the sense that it has no effect on the input and can be removed from the pre-processing process.

Proposition 3.2. *Let* Π^1 *be defined by:*

$$\Pi^1 = R(\Omega^1 \oplus K', K) \tag{3.5}$$

where $R(X, K)$ *applies an AES encryption round on the state* X *and Round key* $\boldsymbol{K}$*.* K' *is a state, where four bytes of* K' *is set to the guessed bytes of* K *(those xored with the active bytes of* Ω^1*) and the other bytes are set to zero. Then* Π^1*-Set is a* Λ^1*-Set.*

Proof. The proof is straight forward. If guessed bytes in K' of the 4 active bytes are correct, the pre-whitening step is neutralized by the xor with K', and the result is an Ω^1-Set. Then SB, SR and MC will remove the effect of SB^{-1}, SR^{-1} and MC^{-1} respectively, thus the result will be a Λ^1-Set. □

Figure 3.2 illustrates the difference propagation of a Λ^1-Set after applying an AES decryption round followed by the pre-whitening step and by an AES encryption round (here $\mathrm{S}_{0,0}$ is assumed to be the active byte).

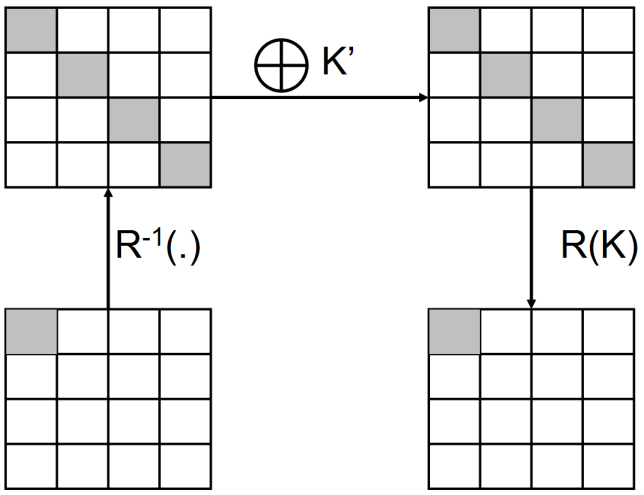


Figure 3.2: An example of applying the pre-processing step on a Λ^1-Set, when $\mathrm{S}_{0,0}$ is the active byte (with pre-whitening).

Proposition 3.3. *Let* Δ^1 *be defined by:*

$$\Delta^1 = R(\Omega^1, 0) \tag{3.6}$$

where $R(X, K)$ *applies an AES encryption round on the state* X *and Round key* $\boldsymbol{K}$*. Then* Δ^1*-Set is a* Λ^1*-Set.*

Proof. The proof is straight forward. SB, SR and MC will remove the effect of SB^{-1}, SR^{-1} and MC^{-1} respectively, thus the result will be a Λ^1-Set. AK will not change the difference, thus the result is still a Λ^1-Set. □

Figure 3.3 illustrates the difference propagation of a Λ^1-Set after applying an AES decryption round followed by an AES encryption round (here $\mathrm{S}_{0,0}$ is assumed to be the active byte).

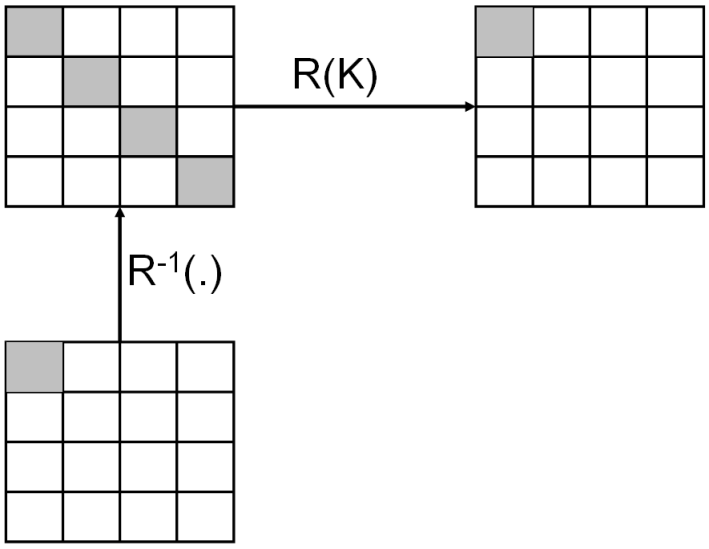


Figure 3.3: An example of applying the pre-processing step on a Λ^1-Set, when $S_{0,0}$ is the active byte (without pre-whitening).

Preparing an Active Column

Let a Δ^4, denotes a set of 2^{32} states, where exactly 4 bytes are active and the positions of the active bytes after applying SR will map to one column.

Proposition 3.4. *Let* Γ^4 *be defined by:*

$$\Gamma^4 = R(R^{-1}(\Delta^4, 0), K) \tag{3.7}$$

where R(X,K)/$R^{-1}(X, K)$ *applies an AES encryption/decryption round on the state* $\boldsymbol{X}$ *and Round key* $\boldsymbol{K}$*. Then* Γ^4*-Set is a* Δ^4*-Set.*

Proof. The proof is straight forward. SB, SR and MC will remove the effect of SB^{-1}, SR^{-1} and MC^{-1} respectively, thus the result will be a Δ^4-Set. AK will not change the difference, thus the result is still a Δ^4-Set. □

Figure 3.4 illustrates the difference propagation of a Δ^4-Set after applying an AES decryption round followed by an AES encryption round (here $S_{0,0}$, $S_{1,1}$, $S_{2,2}$ and $S_{3,3}$ are assumed to be the active byte).

3.2.2 Pushup Attack

The Pushup attack hosts a chosen ciphertext attack, where it prepares the chosen ciphertext for that attack. Figure 3.5 gives an overview on how it works, where:

1. In Figure 3.5(a), an **n** round chosen ciphertext attack is presented that accepts the set **{X}** of chosen ciphertexts to calculate the Key **K**.

2. Figure 3.5(b) presents the pre-processing step of the Pushup attack, where **{X}** is transformed to **{Y}**.

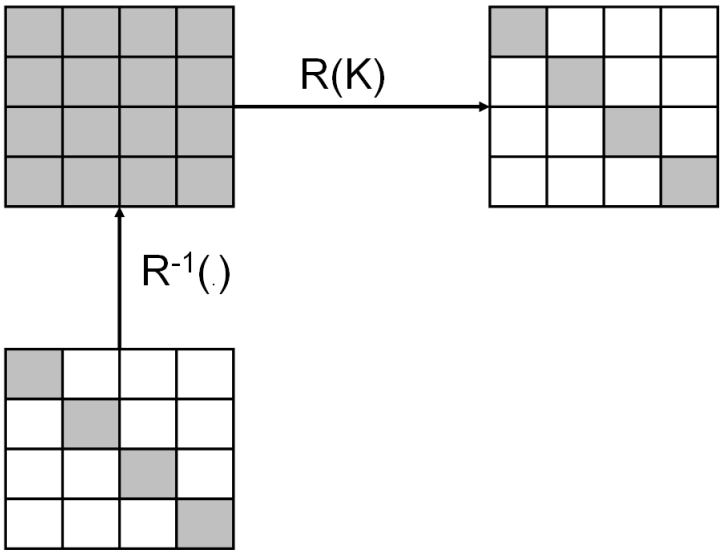


Figure 3.4: An example of applying the pre-processing step on a Δ^4-Set, when $S_{0,0}$, $S_{1,1}$, $S_{2,2}$ and $S_{3,3}$ are the active bytes.

3. Figure 3.5(c) presents the Pushup attack, where after decrypting **{Y}** with **r** rounds, the result is **{X'}**, which should be equivalent to **{X}**, in the sense that applying the n round chosen ciphertext attack on **{X'}** will recover **K**. Thus, the number of rounds attacked by the chosen ciphertext attack is increased to **(n+r)** rounds.

3.2.3 Sandwich Attack

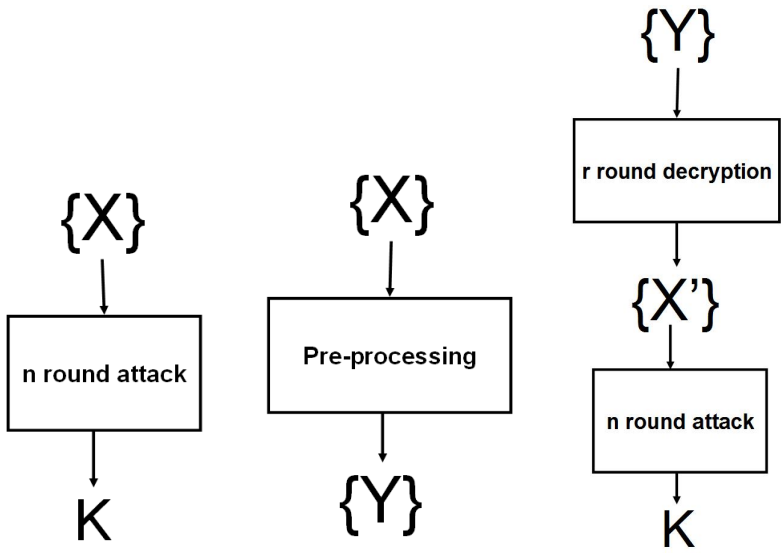
Sandwich attacks hosts either a chosen plaintext-adaptive chosen ciphertext or chosen ciphertext-adaptive chosen plaintext attack. The idea is to apply the concepts of the Pushdown and Pushup attacks on that attack, where the chosen plaintext/ciphertext is pre-processed to bypass some encryption/decryption rounds.

3.3 Pushdown Attacks on AES

The pre-processing of the chosen plaintext in Sect. 3.2.1 is used to deploy the Pushdown attacks on AES.

3.3.1 Pushdown-Square-5

Pushdown-Square-5 attack is based on Square-4 attack, where an Ω^1 is inputted to a 5-round AES, and after applying the first AES round, the input to the second round is now a Λ^1-set (refer to Proposition 3.2), from here Square-4 attack can be applied. Note that whenever a Λ^1-set is needed to be calculated



(a) n round attack (b) Pre-processing (c) Pushup attack

Figure 3.5: Overview on the Pushup attack.

the corresponding Ω^1-set is calculated instead. The complexity of Pushdown-Square-5 is (2^9 chosen plaintext and 2^{40} cipher executions), as 32-bit of the key are needed to be guessed. To the best of the author's knowledge, this attack requires the least amount of chosen plaintext in the literature, to attack 5-round AES.

3.3.2 Pushdown-Square-6

Pushdown-Square-6 attack is based on Square-5 attack, where an Ω^1 is inputted to a 6-round AES, and after applying the first AES round, the input to the second round is now a Λ^1-set (refer to Proposition 3.2), from here Square-5 attack can be applied. Note that whenever a Λ^1-set is needed to be calculated the corresponding Ω^1-set is calculated instead. The complexity of Pushdown-Square-6 is about (5×2^8 chosen plaintext and 2^{72} cipher executions). To the best of the author's knowledge, this attack requires the least amount of chosen plaintext in the literature, to attack 6-round AES.

3.3.3 Pushdown-Square-5*

Pushdown-Square-5* attack is based on Square-4 attack, where an Ω^1 is inputted to a 5-round AES (where the pre-whitening process is omitted), and after applying the first AES round, the input to the second round is now a Λ^1-set (refer to Proposition 3.3), from here Square-4 attack can be applied. Note that whenever a Λ^1-set is needed to be calculated the corresponding Ω^1-set is calculated instead. The complexity of Pushdown-Square-5* is the same as that of Square-4 (2^9 chosen plaintext and 2^9 cipher executions). This attack will be used to break SCC-128 (refer to Sect. 6.8.1).

3.3.4 Pushdown-PartialSums-7*

Pushdown-PartialSums-7* is based on PartialSums-6 attack, where a Γ^4 is inputted to a 7-round AES (where the pre-whitening process is omitted), and after applying the first AES round, the input to the second round is now a Δ^4-set (refer to Proposition 3.4), and the PartialSums-6 attack can be applied on the last 6 rounds. Note that whenever a Δ^4-set is needed to be calculated the corresponding Γ^4-set is calculated instead. The complexity of Pushdown-PartialSums-7* is almost the same as that of PartialSums-6 (6×2^{32} chosen plaintexts, and 2^{44} cipher executions). This attack will be used to break SCC-256 (refer to Sect. 6.8.2).

3.4 Attacks on Key Schedule

Table 3.3 and table 3.4 summarize the complexities of some Related-key attacks on AES-192 and AES-256, where RK refers to Related-Key, CP refers to Chosen Plaintext and the time complexity is measured in encryption units.

3.4.1 Related-key Impossible Differential Attack

Related-key attacks [12] allow an adversary to obtain plaintext-ciphertext pairs by using related (but unknown) keys. The adversary first searches for possible weaknesses of the encryption and key schedule algorithms then chooses appropriate relation between keys and makes two encryptions using the related-keys expecting to derive the unknown key information. Differential cryptanalysis analyzes the involvement of the difference between a pair of plaintexts in the following round outputs in an iterated cipher. Related-key differential attack [90] combines the above two cryptanalytic techniques together, and it studies the development of differences in two encryptions under two related-keys. Furthermore, impossible differential attacks [19] use differentials that hold with probability 0 (or non-existing differentials) to eliminate wrong key material and leave the right key candidate. In this case, the combined attack is called related-key impossible differential attack.

If the expanded keys are viewed as a sequence of 32-bit words, then the key schedule of AES-192 applies a non-linear transformation once every six words, whereas the key schedules of AES-128 and AES-256 apply non-linear transformations once every four words. This property brings better and longer related-key differentials of AES-192, so directly makes AES-192 more susceptible to related-key attacks than AES-128 and AES-256. In the last few years, the security of AES-192 against related-key attacks has drawn much attention from cryptology researchers [17, 18, 78, 83]. In [83], Jakimoski et al. presented related-key impossible differential attacks on 7- and 8-round AES-192. Following the work of [83], Biham et al. [18] gave several new related-key impossible differential attacks also on 7- and 8-round AES-192, which substantially improved the data and time complexity of those in [83].

3.4.2 Related Key Rectangle Attack

Related Key Rectangle Attack [95, 78, 17] combines the rectangle [16] and related-key attacks by applying the rectangle attack to the cipher with different,

Table 3.3: Some related-key attacks on AES-192.

Number of Keys	Source	Number of	Data Complexity	Time Complexity	Attack Type
256	Ref. [17]	9*	2^{86} RK-CP	2^{125}	RK Rectangle
64	Ref. [94]	9**	2^{85} RK-CP	2^{182}	RK Rectangle
4	Ref. [78]	8	$2^{86.5}$ RK-CP	$2^{86.5}$	RK Rectangle
2	Ref. [94]	8	2^{94} RK-CP	2^{120}	RK Rectangle
256	Ref. [94]	10	2^{125} RK-CP	2^{182}	RK Rectangle
64	Ref. [94]	10	2^{124} RK-CP	2^{183}	RK Rectangle
2	Ref. [83]	7	2^{111} RK-CP	2^{116}	RK Imp. Diff
2	Ref. [83]	8	2^{88} RK-CP	2^{183}	RK Imp. Diff
32	Ref. [18]	7	2^{56} RK-CP	2^{94}	RK Imp. Diff
32	Ref. [18]	8	2^{116} RK-CP	2^{134}	RK Imp. Diff
32	Ref. [18]	8	2^{92} RK-CP	2^{159}	RK Imp. Diff
32	Ref. [18]	8	$2^{68.5}$ RK-CP	2^{184}	RK Imp. Diff
2	Ref. [161]	7	2^{52} RK-CP	2^{80}	RK Imp. Diff
2	Ref. [161]	8	$2^{64.5}$ RK-CP	2^{177}	RK Imp. Diff
2	Ref. [161]	8	2^{88} RK-CP	2^{153}	RK Imp. Diff
2	Ref. [161]	8	2^{112} RK-CP	2^{136}	RK Imp. Diff
2	Ref. [161]	7	2^{37} RK-CP	2^{145}	RK Diff
* Attack with some flaws, ** corrected flaws.					

Table 3.4: Some related-key attacks on AES-256.

Number of Keys	Source	Number of	Data Complexity	Time Complexity	Attack Type
2	Ref. [160]	7	2^{52} RK-CP	2^{87}	RK Imp. Diff
2	Ref. [160]	8	2^{53} RK-CP	2^{215}	RK Imp. Diff
2	Ref. [160]	8	2^{64} RK-CP	2^{191}	RK Imp. Diff
2	Ref. [160]	8	2^{88} RK-CP	2^{167}	RK Imp. Diff
2	Ref. [160]	8	2^{112} RK-CP	2^{143}	RK Imp. Diff
256	Ref. [17]	10	$2^{114.9}$ RK-CP	$2^{171.8}$	RK Rectangle
4	Ref. [94]	9	2^{99} RK-CP	2^{120}	RK Rectangle
64	Ref. [94]	10	$2^{113.9}$ RK-CP	$2^{172.8}$	RK Rectangle

but related unknown keys: [95, 78, 17] show how to apply the rectangle attack with 2, 4 and more than 4 related-keys, and show that this kind of attack can be applied to AES-192 and AES-256. The related-key rectangle attack is based on two consecutive related-key differentials with relatively high probabilities which are independent of each other [94].

3.5 Proposed Enhanced AES Key Schedule

Compared to the cipher itself, AES key schedule appears to be more of an ad-hoc design. It has a much slower diffusion structure than the cipher, and contains relatively few non-linear elements [61]. To be able to compute the rounds key on-the-fly, Rijndael's key schedule has an interesting property that is if you have a round key you can calculate all the other round keys and retrieve the encryption key. Although this property increases the key agility of Rijndael, it has been used in many theoretical and side channel attacks. For example in the square attack [36], by recovering the round key of the last round, one can recover all the other round keys including the encryption key. In [123, 122, 25] cache timing attacks have been used to recover either the first or the last round key, which were used to extract the encryption key. To simplify the analysis and to enhance the security of the proposed models, a modified key schedule of AES is proposed. This key schedule eliminates related-key attacks along with some other attacks on AES.

3.5.1 Rijndael Key Schedule Classification

In [29], the authors introduced a classification scheme for iterative block ciphers based on their key schedules. This scheme creates two categories of ciphers based on whether or not knowledge of a round key generated by the key schedule reveals any information about other round keys or the encryption key. Those that do, fall into *Category 1* and those that do not, fall into *Category 2*. Each of these categories is further subdivided into three types: A, B and C. Theses categories are:

1A: Encryption Key is used without any modification.

1B: Easily reversed operations applied on the Encryption Key.

1C: Round keys are calculated from earlier round keys.

2A: Encryption Key bits are partially used in the derivation process for the round keys.

2B: All Encryption Key bits are used in the derivation process.

2C: Round keys are independent of each other and the sum of all round keys equals the length of the Encryption Key.

In [30], the key schedules of AES candidates were classified. Rijndael's key schedule was classified to be 1C, where the knowledge of a round key reveals bits of other round keys or the encryption key after some simple arithmetic operations or function inversions.

3.5.2 Proposed Key Schedule for AES

The goal is to increase the security of AES' key schedule, by increasing its classification number, namely to 2B, where all encryption key bits are used in the determination of all round keys, thus maximizing the entropy of the round keys and by knowing one of the round keys no information about any other round keys or the encryption key can be extracted. The idea is to use the current AES implementation to increase the security of AES' key schedule, precisely AES in counter mode [113] is used to generate AES' expanded key. The listing of the proposal is in table 3.5, where *Counter* is a 128-bit block, *Expand-Key* expands the encryption key using AES key schedule routine and *Encrypt-AES* performs AES encryption function. If the equivalent inverse cipher construction [38] is used for decryption, then the expanded decryption key can be generated by applying InvMixColumns to all round keys except that for first and the last rounds.

Table 3.5: The proposed AES' key schedule.

```
Enhanced-Expanded-Key(MasterKey)
    ExpandedKey=Expand-Key(MasterKey)
    Counter= 0
    for i=0 to n-1
        OUT_i=Encrypt-AES(ExpandedKey,Counter)
        Counter=Counter+1
    end for
return OUT
```

It is straight forward to see that the security of the proposed key schedule is inherited from that of AES. In other words, if there is a method to calculate a round key/encryption key from another round key, this implies that AES is broken. The proposed key schedule protects the current AES implementation from many attacks like related-key attacks (refer to Sect. 3.4) and some cache timing attacks [123, 122, 25]. It is worth mentioning that this method increases the time complexity of many attacks, even the exhaustive key search attack, in the sense that to try a key 11 to 15 AES encryption calls are needed.

The only drawback of the proposed solution is its performance, as for example to generate the expanded keys of AES-128 the cipher should be called 11 times, and for AES-256 it should be called 15 times. So this method is not suitable for the applications that use on-the-fly round keys computations, and is more suitable to the applications that do not change the expanded key frequently (e.g. the proposed encryption models in Chapter 4, where the primary key has a long lifetime).

Intel has introduced a new set of instructions that will be available in all Intel processors as of the 2010 generation [70]. These instructions will increase the performance of AES, for example the expected speed of one block encryption (16 bytes) using CBC mode and AES with 128-bit is about 65 cycles. While for the parallel modes (ECB, CTR or CBC decryption), using AES instructions and loop reversing (loop unrolling) software implementation, the achieved throughput is around 24 cycles/block (on a Quad-Core possessor). This means the expected performance of the proposed key schedule on a Quad-Core proces-

sor, is about 280 cycles for AES-128 and about 480 cycles for AES-256, which is considered very fast.

In the rest of this thesis, it is assumed that the Enhanced AES Key Schedule is used instead of the original AES key schedule.

3.5.3 Proposed Generalized Key Schedule

The proposal can be generalized to be used by any block cipher key schedule. The listing of the proposal is in table 3.5, where *BlockCipher-Expand-Key* is the key schedule of a secure block cipher, *Encrypt-BlockCipher* is the encryption function of the secure block cipher, n is the number of rounds of the cipher and *Counter* is a block of the cipher's block size.

Table 3.6: The proposed generalized key schedule.

```
Generalized-Enhanced-Expanded-Key(MasterKey)
    ExpandedKey=BlockCipher-Expand-Key(MasterKey)
    Counter= 0
    for i=0 to n-1
        OUT_i=Encrypt-BlockCipher(ExpandedKey,Counter)
        Counter=Counter+1
    end for
return OUT
```

3.6 Side Channel Attacks

Side Channel attacks do not attack the cipher itself, but rather how it is implemented. AES has been a subject to intensive mathematical cryptanalysis. But to the best of the author's knowledge, AES has not been broken using analytical attacks. Unfortunately, there are a couple of successful side channel attacks on AES, like the attacks published in [123, 122, 25] to name a few. The discussion of side channel security is beyond the scope of this thesis.

3.7 Summary

In this chapter, a new idea is presented that can increase the strength of some chosen plaintext attacks. The idea is to modify the chosen plaintext, in such a way that after applying r encryption rounds, the original/equivalent chosen plaintext is produced. This idea can increase the strength of an (n) round attack to an (n + r) round attack, as the first r rounds are bypassed. Based on this idea the Pushdown attack is deployed which results in 5- and 6-round attacks on AES. To the best of the author's knowledge, these attacks require the least amount of chosen plaintext in the literature. It is also discussed: how the idea of preprocessing can be implemented on chosen ciphertext, chosen plaintext-adaptive chosen ciphertext and chosen ciphertext-adaptive chosen plaintext attacks.

A new key schedule is proposed to enhance the security of AES, by using AES in counter mode to generate the round keys. It is straightforward to see that the security of the proposed key schedule is inherited from that of AES.

In other words, if there is a method to calculate a round/encryption key from another round key, then AES is broken. The proposed key schedule protects the current AES implementation from many attacks like related-key and some cache timing attacks. It is worth mentioning that this method increases the time complexity of many attacks, even the exhaustive key search attack (in the sense that to try a key 11 to 15 encryption steps are needed). A generalized secure block ciphers' key schedule is also proposed.

Chapter 4

Proposed Encryption Models

4.1 State of the Art Encryption Models

4.1.1 Classical Encryption Model

Figure 4.1 sketches the classical encryption model, where only the encryption key controls how the plaintext will be encrypted.

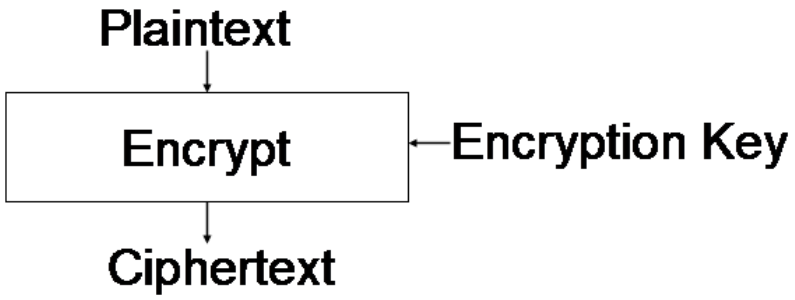


Figure 4.1: Classical Encryption Model.

4.1.2 Tweakable Block Ciphers

Tweakable Block Ciphers are a new cryptographic primitive, presented in [105]. Such a cipher has not only the usual inputs (message and key) but also a third input, the "tweak". The tweak serves much the same purpose that an initialization vector does for CBC mode or that a nonce does for OCB mode [142]. The tweak, along with the key, selects the permutation computed by the cipher. Changing tweaks should be sufficiently lightweight (compared with the expensive key setup operation). In [105], two constructions 4.1 and 4.2 were proposed, where $E'_K(T, M)$ is the tweakable block cipher encryption function, which encrypts **M** using **K** as the encryption key and **T** as the tweak, **h** is a hash function and $E_K(P)$ encrypts **P** using **K** as its encryption key.

The first construction (4.1) is considered slow, as it uses two encryption calls. The second construction (4.2) is more efficient, when **h** is a fast hash function. Fast hash functions like generalized division [148], UMAC/UHASH [23], hash127 [11] and a DFC-style decorrelation module [67], can be used.

$$E'_K(T,P) = E_K(T \oplus E_K(P)) \tag{4.1}$$

$$E'_K(T,P) = E_K(P \oplus h(T)) \oplus h(T) \tag{4.2}$$

After the introduction of tweakable block ciphers, many new tweakable modes of operation have been developed. Most of these modes are dedicated to disk encryption applications. Examples of these modes are LRW [82], XTS [141], EME [74], EME* [71], XCB [114], TET [72], CMC [75], ABL4 [115].

Figure 4.2 sketches the tweakable encryption model, where the encryption key together with the tweak control how the plaintext should be encrypted.

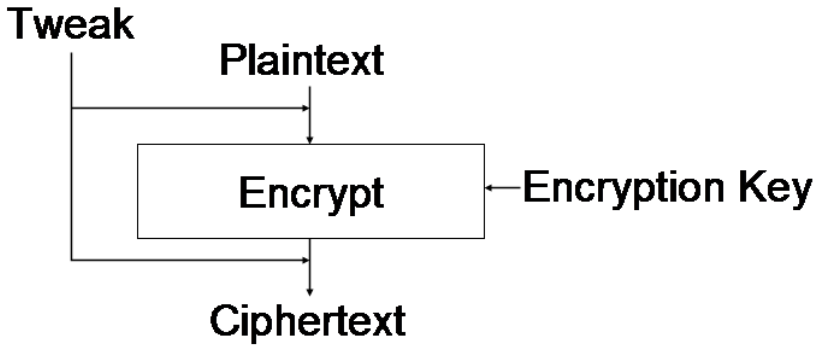


Figure 4.2: Tweakable Encryption Model.

4.2 General Scheme of the Proposed Models

All the proposed models share a main idea, which is splitting the encryption key into a primary and a secondary key. The secondary key together with the primary key, are used to determine how the plaintext will be encrypted. The main functionality of the secondary key is to change the way the block cipher behaves, in other words by encrypting two identical plaintexts with the same primary key and two different secondary keys, the results will be two different ciphertexts.

Figure 4.3 sketches the general scheme of the proposed encryption models, where the encryption key is divided into two keys; the primary key which is used to generate the expanded key of the cipher, and the secondary key which is used to modify the expanded key of the cipher. As the secondary key modifies the expanded key of the cipher; it must be kept secret as it is part of the encryption key and the adversary must not have control over its value.

In tweakable block ciphers, the tweak manipulates the input and the output of the cipher (where the block cipher is treated as a black box). The proposed models treat a block cipher as a white box, where the secondary key manipulates

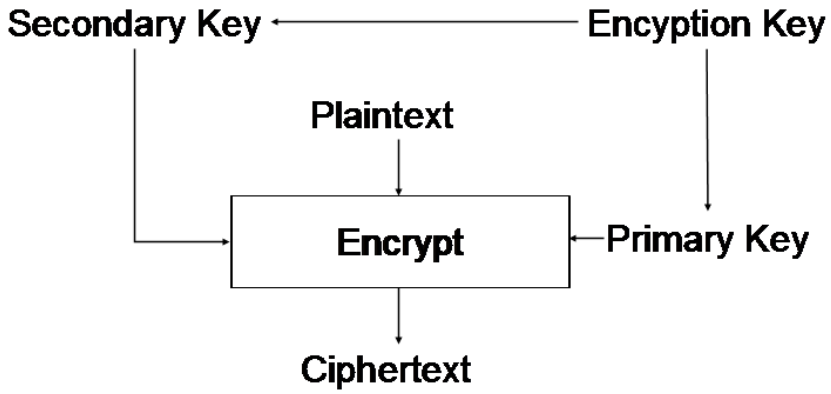


Figure 4.3: General scheme of the proposed encryption models.

the cipher's expanded key. Three main models are proposed: Dynamic Substitution Model (DSM), Dynamic Injection Model (DIM) and Dynamic Permutation Model (DPM). In addition to these main models, the possibility to construct hybrid models using different combinations of these models is discussed.

Note that: the secondary key manipulates the cipher's expanded key (the cipher's inner state), thus the security of the models relies on the security of the secondary key together with that of the primary key. It can be thought that the secondary key together with the primary key construct the encryption key.

4.3 Terminologies and Definitions

4.3.1 Terminologies

These terminologies are used to describe the proposed encryption models:

P: is the input plaintext.

E: is a block cipher.

EK: is the expanded encryption key of **E**.

m: in DSM is the number of words in **EK** that will be replaced, and in DIM is the number of the transformation functions.

SK: is the secondary key; it is structured as an array of words in DSM, and it is structured as an array that holds the keys for the transformation functions in DIM.

F: is an array that holds **m** keyed transformation functions used in the encryption process of DIM.

I: is an array of indices that determines which words of **EK** will be replaced in DSM. In DIM, **I** determines where the **F** functions should be injected and the values of **I** must be unique and sorted in an ascending order.

C: is the output ciphertext.

C=$\mathbf{E}_{EK}$(P): encrypts **P** with the block cipher **E**, using **EK** as the expanded encryption key and returns the result.

$\mathbf{F}_i$(X,K): applies the ith transformation function on **X** using **K** as its key and returns the result.

$\mathbf{R}_i$(X,EK): applies the ith round of **E** on **X** using the proper round key from **EK** and returns the result, where the proper round key is the part of the expanded key that is consumed by the ith round function.

round(X,K): applies an AES encryption round on **X**, using **K** as the round key and returns the result.

final(X,K): applies the final AES encryption round on **X**, using **K** as the round key and returns the result.

Substitute(EK,K,i): replaces the i^{th} 128-bit of **EK** with **K** (Note that: the first round of AES is round zero and it is the pre-whitening process).

len(X): returns the size of **X** in bits.

4.3.2 Definitions

A modified cipher that uses the proposed models is called a two-key block cipher, where (4.3) presents encrypting the plaintext **P** using the block cipher **E**, where **PK** is the primary key and **SK** is the secondary key to produce the ciphertext **C**. The decryption process is represented by (4.4).

$$C = E(PK, SK)(P) \tag{4.3}$$

$$P = E^{-1}(PK, SK)(C) \tag{4.4}$$

Variability is the ability of a two-key block cipher to vary its output depending on its secondary key, more formally:

Variability: For two-key block ciphers if C1=E(PK,SK1)(P) and C2=E(PK,SK2)(P) then $C1 \neq C2$ as long as $SK1 \neq SK2$.

Strong variability is the ability of a two-key block cipher to protect its secondary key, in the sense that by knowing a set of plaintext/ciphertext pairs, it is hard to recover the secondary key, more formally:

Strong Variability: By encrypting the same plaintext with different instances of a two-key block cipher using the same primary key and different secondary keys, it is hard to recover the secondary key (totally or partially).

Differentiability is the ability to have strong variability, even if the primary key is shared, more formally:

Differentiability: If the primary key is shared among (m) parties and each party has its unique secondary key. A two-key block cipher E(PK,SK) offers differentiability, if by knowing the secondary key of a party, the secondary keys of the other parties cannot be easily recovered.

4.4 Dynamic Substitution Model (DSM)

DSM is a model that can provide a block cipher with a variable length secondary key. The secondary key is used to replace some words of the cipher's expanded key. Two extra inputs are introduced by DSM to the classical encryption model. The first input (**SK**) is the secondary key that will substitute some words in the expanded key of the cipher, the second input (**I**) holds the indices which specify which words in the expanded key will be substituted. Note that the substituted words can be of any size. The listing of Encrypt-DSM, which is used for encryption in DSM, is in table 4.1.

Table 4.1: Encrypt-DSM function.

```
Encrypt-DSM(P,E,EK,SK[m],I[m])
    for i=0 to m-1
        EK[I[i]]=SK[i]
    end for
    C=E_EK(P)
return C
```

Encrypt-DSM executes as follows:

- The words of the secondary key **SK** replace certain words in **EK** (determined by the index array **I**).
- The updated **EK** is used as the new expanded key for the encryption function, which encrypts the input plaintext (**P**) to produce the ciphertext (**C**).

4.4.1 DS-AES

In order to demonstrate DSM, the Dynamic Substitution AES (DS-AES) is proposed. The listing of DS-AES encryption function is found in table 4.2, where:

- **EK** is the expanded AES encryption key, and it is an array of 32-bit words.
- **C=Encrypt-AES(P,EK)** encrypts **P** with AES using **EK** as the expanded encryption key and returns the ciphertext in **C**.

4.5 Static Substitution Model (SSM)

SSM provides a fixed length secondary key. It is a special case of DSM, where the secondary key length and the index array **I** are determined in the design time. SSM can build more efficient implementations than DSM. SSM is used to construct three variants of AES. These variants are AESS1, AESS2 and AES2S.

Table 4.2: DS-AES encryption function.

```
Encrypt-DS-AES(P,EK,SK[m],I[m])
    for i=0 to m-1
        EK[I[i]]=SK[i]
    end for
    C=Encrypt-AES(P,EK)
return C
```

4.5.1 AESS1, AESS2 and AES2S

Table 4.3 presents the encryption functions of AESS1 and AESS2, where **SK** is the secondary key, **FR** and **SR** are round numbers. AESS1 accepts 128-bit secondary key. **FR** (the round number) is determined depending on the expanded key length.

AESS2 accepts 256-bit secondary key and depending on the expanded key length **FR** and **SR** (the round numbers, where **SR** $>$ **FR**) are determined.

Table 4.3: AESS1 and AESS2 encryption functions.

```
Encrypt-AESS1(P,EK,SK)
if(len(EK)==1408)
    FR=5

else
    FR=7

end if
for i=0 to 3
    EK[4 × FR + i ]=SK[i]

end for
    C=Encrypt-AES(P,EK)
return C
```

```
Encrypt-AESS2(P,EK,SK)
if(len(EK)==1408)
    FR=3
    SR=7
else
    FR=4
    SR=10
end if
for i=0 to 3
    EK[4 × FR + i ]=SK[i]
    EK[4 × SR + i ]=SK[4+i]
end for
    C=Encrypt-AES(P,EK)
return C
```

AES2S is a variant of AES that accepts 256-bit secondary key. The listing of AES2S is found in table 4.4. In AES2S, two round keys are replaced:

1. The round key of the pre-whitening round is replaced with the first 128-bit of **SK**.

2. The round key of the xth round is replaced with the last 128-bit of **SK**.

4.5.2 Advantages of DSM and SSM

The advantages of DSM and SSM are:

Table 4.4: AES2S encryption function.

```
Encrypt-AES2S(P,EK,SK)
if(len(EK)=1408)
      x=7
else
      x=10
end if
Substitute( EK , SK , 0 )
Substitute( EK , SK+4 , x )
C=Encrypt-AES( P , EK )
return C
```

Generality: DSM and SSM can be applied to any block cipher, with expanded key.

High throughput: DSM and SSM have almost no overhead and the block cipher will execute with high speed.

Low memory consumption: Only the secondary key is needed to be stored per instantiation, when the primary key is shared among n instances (e.g. the proposed network encryption schemes in Chapter 5).

4.6 Dynamic Injection Model (DIM)

DIM is a model that can provide a block cipher with a variable length secondary key. The secondary key is consumed by some keyed transformation functions that are injected into the cipher. A transformation function can vary in strength from a simple xor function to a complete cipher. The transformation functions modify the input and/or the output of some of the cipher's round functions (using parts of the secondary key as their key). Three extra inputs are added by DIM to the classical encryption model. The first input (**F**) is an array that holds the transformation functions, the second input (**SK**) holds the secondary key that will be consumed by the transformation functions, the third input (**I**) holds the indices, which determine where the transformation functions should be injected.

The listing of Encrypt-DIM, which is used for encryption in DIM is in table 4.5.

Encrypt-DIM executes as follows:

1. Before applying the first round of the cipher, the plaintext **P** is copied to the text **X** and Encrypt-DIM checks if the first transformation function should update the text **X** and if this is the case **X** is modified using the first transformation function and its corresponding part of the secondary key.

2. Encrypt-DIM applies a round function of **E** (using the proper round key from the expanded encryption key **EK**) and updates **X**, where the proper

Table 4.5: Encrypt-DIM function.

```
Encrypt-DIM(P,E,EK,F[m],SK[m],I[m])
    X=P
    t=0
    If (I[0]==0) then
        X= F_0(X,SK_0)
        t++
    End if

    For i=1 to n
        X= R_i(X,EK)
        If(I[t]==i) then
            X= F_t(X,SK_t)
            t++
        End if
    End for
return X
```

round key is the part of the expanded key that is consumed by this round function.

3. Encrypt-DIM checks if the next transformation function should update the text **X** and if this is the case **X** is updated.

4. The last two steps are repeated another **n-1** times (where **n** is the number of rounds of the cipher **E**).

5. Encrypt-DIM returns **X** as the ciphertext.

4.6.1 Transformation Functions

The strength of transformation functions can vary from a simple xor function to a complete cipher. But care should be considered, as these functions are supposed to add strength to the cipher or in the modest case not to weaken the cipher. Functions that reverse the effect of the cipher's round functions (or even part of them) must be avoided.

The total number of bits consumed by these functions is the secondary key length. Each transformation function can modify the entire input or only part of it. Modifying the same input more than once is also possible, as more than one function can be injected in the same position; in this case they are treated as one function. But care should be taken that these functions do not cancel any part of the original cipher's round functions nor form a group (e.g. injecting two xor functions with K1 and K2 as their key, is equivalent to injecting only one xor function with K3, where K3 = K1 $\oplus$ K2).

4.6.2 DI-AES

In order to demonstrate DIM model, the Dynamic Injection AES (DI-AES) is presented. The listing of DI-AES encryption function is found in table 4.6,

where **n** is the number of rounds: typically 10, 12 and 14 for the primary key of size 128-bit, 192-bit and 256-bit respectively and **xor(X,Y)** : xors **X** with the first 128-bit of **Y** and returns the result.

Table 4.6: DI-AES encryption function.

```
Encrypt-DI-AES(X,EK,F[m],SK[m],I[m])
    t=0
    If (I[0]==0) then
        X= F_0(X,SK_0)
        t++
    End if

    X= xor(X,EK)
    For i=1 to n - 1
        If(I[t]==i) then
            X= F_t(X,SK_t)
            t++
        End if
        X= round(X,EK + 4 × i)
    End for

    If(I[t]==n-1) then
        X= F_t(X,SK_t)
        t++
    End if

    X= final(X,EK + 4 × (n-1))
    If(I[t]==n) then
        X= F_t(X,SK_t)
    End if
return X
```

After presenting DI-AES, here are some examples of its use:

1. If a secondary key of length 128-bit is needed, the direct way is to inject one function that accepts 128-bit key (e.g. an AES round function or an xor function) in the middle of the cipher (i.e. I[0]= n/2). Another solution is to inject two functions in two different positions, each function consumes 64-bit.

2. To construct AESX (DESX [140] like cipher), the secondary key should be 256-bit and two xor functions should be injected, one at the beginning and the other at the end (i.e. I[0]=0 and I[1]=n).

3. If for a certain application a 140-bit secondary key is needed, one way is to inject the DI-AES with one AES round (e.g. I[0]=n/2) and inject an xor function (that modifies only 12-bit) in another place (e.g. I[1]= n/3).

4.7 Static Injection Model (SIM)

SIM provides a fixed length secondary key. It is a special case of DIM, where the keyed transformation functions and their positions are determined in the design time. SIM can build more efficient implementations than DIM. SIM is used to construct three variants of AES. These variants are AESI1, AESI2 and AES2I.

4.7.1 AESI1, AESI2 and AES2I

AESI1 accepts 128-/256-bit primary key and 128-bit secondary key. In AESI1, AES is injected with an extra AES round. The choice was to inject AES after the $5^{th}/7^{th}$ round.

AESI2 accepts 128-/256-bit primary key and 256-bit secondary key. In AESI2, AES is injected with two xor functions. The choice was to inject AES after the $3^{rd}/4^{th}$ and $7^{th}/10^{th}$ rounds.

AES2I accepts 128-/256-bit primary key and 256-bit secondary key. In AES2I, AES is injected with two xor functions. The choice was to inject AES before the pre-whitening process and after the last round.

Table 4.7 and table 4.8 present the encryption functions of AESI1 and AESI2, where **SK** is the secondary key. Table 4.9 presents the encryption functions of AES2I, where **SK1** and **SK2** are 128-bit keys that construct the secondary key.

Table 4.7: AESI1 and AESI2 encryption functions (128-bit version).

Encrypt-AESI1(X,EK,SK)	Encrypt-AESI2(X,EK,SK)

xor(X,EK)
for i=1 to 5
  round(X,EK+4 × i)
end for
round(X,SK)
for i=6 to 9
  round(X,EK+4 × i)
end for
final(X,EK+10 × 4)
return X
	xor(X,EK)
	for i=1 to 3
	  round(X,EK+4 × i)
	end for
	xor(X,SK)
	for i=4 to 7
	  round(X,EK+4 × i)
	end for
	xor(X,SK+4)
	for i=8 to 9
	  round(X,EK+4 × i)
	end for
	final(X,EK+10 × 4)
	return X

4.7.2 Advantages of DIM and SIM

Generality: DIM and SIM can be applied to any iterative block cipher.

Table 4.8: AESI1 and AESI2 encryption functions (256-bit version).

Encrypt-AESI1(X,EK,SK)	Encrypt-AESI2(X,EK,SK)

xor(X,EK)
for i=1 to 7
  round(X,EK+4 × i)
end for
round(X,SK)
for i=8 to 13
  round(X,EK+4 × i)
end for
final(X,EK+14 × 4)
return X
	xor(X,EK)
	for i=1 to 4
	  round(X,EK+4 × i)
	end for
	xor(X,SK)
	for i=5 to 10
	  round(X,EK+4 × i)
	end for
	xor(X,SK+4)
	for i=11 to 13
	  round(X,EK+4 × i)
	end for
	final(X,EK+14 × 4)
	return X

Table 4.9: AES2I encryption function.

Encrypt-AES2I(X,EK,SK1,SK2)
xor(X,SK1) X=Encrypt-AES(X,EK) **xor(X,SK2)** return X

High throughput: DIM and SIM can build ciphers' variants with high speed, when a high speed function like an xor functions is used as the transformation function.

Low memory consumption: Only the secondary key is needed to be stored per instantiation, when the primary key is shared among n instances (e.g. the proposed network encryption schemes in Chapter 5).

Security: DIM can build more secure variants of the cipher (e.g. by increasing the number of rounds of that cipher).

4.8 Dynamic Permutation Model (DPM)

DPM is a model that can provide a block cipher with a secondary key. The secondary key is used to permute some words of the cipher's expanded key. An extra input is introduced by DPM to the classical encryption model. This input is the secondary key (**SK**) that will permute some words in the expanded key of the cipher. Figure 4.4 presents an overview of DPM. It works as follows:

1. **Initialization**: where the primary key is expanded using the setup algorithm (usually the encryption key setup routine of the used cipher) to produce the expanded key.
2. **Encryption**:
   - The expanded key together with the secondary key are given to a permutation function, this function uses the secondary key to permute the expanded key, the result is the new expanded key.
   - The encryption function uses the new expanded key (from the previous step) to encrypt the plaintext.

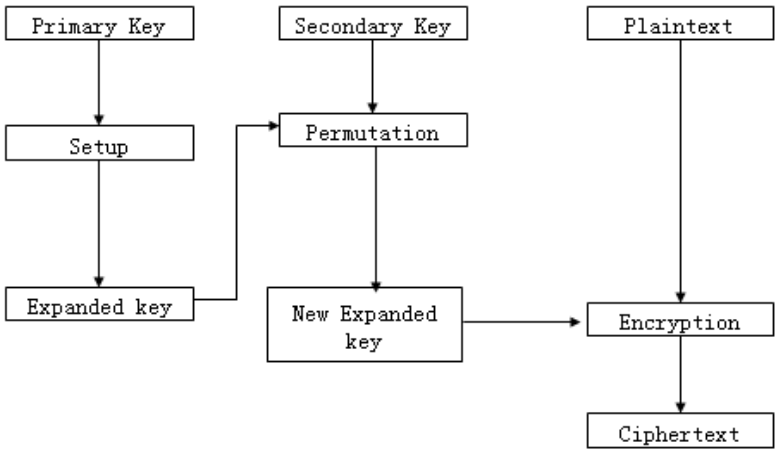


Figure 4.4: Overview of Dynamic Permutation Model.

In the following text, an implementation example on how to build a new AES variant based on DPM is presented.

4.8.1 Dynamic Permutation AES

Dynamic Permutation AES (DP-AES) is a variant of AES-256 (with a secondary key of size 24-bit, it is presented here as an example of implementing DPM). With the fact that AES-256 has 60 words (each with 32-bit) as its expanded key, the generation of all these permutations will be costly. A fast solution is to use a table lookup with permutations. The table size must use acceptable memory resources. The choice was to use a 256 x 24-bit table. So the table entry needs 8-bit and returns a permutation of 8. It uses 24-bit secondary key. The listing of Encrypt-DP-AES, which is used for encryption in the proposed variant, is in table 4.10, where:

Table 4.10: DP-AES encryption function.

```
Encrypt-DP-AES(P, Exkey, SK)
C=xor(P,ExKey)
for i=1 to 4
    C=round(C,ExKey + i × 4)
end for
for i=0 to 2
    C=round1(C,ExKey + (i+5) × 4,SK[i])
    C=round2(C,ExKey + (i+5) × 4,SK[i])
end for
for i=11 to 13
    C=round(C,ExKey + i × 4)
end for
C=final(C,ExKey + 14 × 4)
return In
```

- It takes the plaintext (**P**), the expanded encryption key (**ExKey**) and the secondary key (**SK**) as inputs.
- The first 4 rounds of AES are applied to the plaintext.
- Round1 and round2 functions (which are modified AES rounds) are executed three consecutive times, each time with **SK[i]** as their input.
- Finally it executes three normal AES rounds, the final AES round and returns the ciphertext (**C**).
- The permutation is done for each two rounds together, using the functions round1 and round2.
- The functions round1 and round2 use **SK[i]** as a lookup entry in **MBOX**, which returns a permutation of size eight, they use this permutation to permute their expanded key.
- **MBOX** has the following properties:
    1. The probability for each word to be moved to any of the available eight locations is exactly the same 2^{-3}.

2. Maximum 2 words in each row do not change their positions.
3. At least one word from the left half goes to the right half.
4. The rows in the table are unique.

The values of **MBOX** are presented in Appendix A.

- Example:
  - Let **SK[0]**= 10.
  - Let **MBOX[10]**={7, 5, 3, 6, 2, 4, 0, 1}.
  - Then the indices of **ExKey** that round1 will access are: {27, 25, 23, 26} and those of round2 are: {22, 24, 20, 21}.
  - Notes:
    * For normal AES rounds, round1 and round2 should have accessed the following indices {20, 21, 22, 23} and {24, 25, 26, 27} respectively.
    * The indices of round1 and round2 can be calculated, by adding the values of the MBOX[SK[i]] to the first index of the first normal AES round, where SK[i] is an 8-bit part of the secondary key.

4.9 Security of the Proposed Encryption Models

In the following text, the guidelines to construct secure two-key block ciphers using the proposed models are presented.

4.9.1 The Guidelines for Single Round Modification

This construction can be used to provide variability and strong variability. The guidelines to modify the round key of a single round, where **m=1** and **I[0]=x**, are:

1. *The secondary key MUST be unique, random and not predictable by the adversary*, using a secure cipher like AES in counter mode [113] to generate secondary keys, can achieve this at a low cost.

2. *The secondary key MUST NOT be controlled by the adversary*, by controlling the values of the secondary key, the adversary can introduce a difference in the middle of the cipher, which can lead to new attacks (e.g. the attacks in [51, 50]).

3. $\mu \leq x \leq n - \mu$, *where* $\mu = max(l, d)$, *l is the number of rounds the cipher is resistant to linear cryptanalysis [110] and d is the number of rounds the cipher is resistant to differential cryptanalysis [20].* This constraint is to avoid a successful attack using linear/differential cryptanalysis, where a linear/differential propagation is avoided.

4. *Full confusion and full diffusion are expected in both encryption and decryption directions.* This is to provide strong variability.

5. $x < n - \nu$, *where* ν *is the number of rounds the cipher is resistant to chosen plaintext attacks.* If the adversary can encrypt the same plaintext with the same primary key and different secondary keys, she can introduce a difference in the intermediate encrypted text, this difference can be used in a chosen plaintext attack (if the adversary can choose/find secondary keys to share a certain pattern and mount a chosen plaintext attack).

6. $x > \theta$, *where* θ *is the number of rounds the cipher is resistant to chosen ciphertext attacks.* If the adversary can decrypt the same ciphertext with the same primary key and different secondary keys, she can introduce a difference in the intermediate decrypted text, this difference can be used in a chosen ciphertext attack (if the adversary can choose/find secondary keys to share a certain pattern and mount a chosen ciphertext attack).

7. *The secondary key should modify the entire block.* Modifying a partial block increases the probability of certain attacks, like Square attack [36], where the adversary should find intermediate texts that vary in certain positions and are constant in the other positions.

4.9.2 The Guidelines for Double Round Modification

This construction can be used to provide differentiability together with strong variability. The guidelines to modify two round keys, where **m=2**, **I[0]=x** and **I[1]=y**, are:

1. *The two secondary keys MUST be unique, random and not predictable by the adversary*, using a secure cipher like AES in counter mode [113] to generate secondary keys, can achieve this at a low cost.

2. *The secondary keys MUST NOT be controlled by the adversary*, by controlling the values of the secondary keys, the adversary can introduce a difference in the middle of the cipher, which can lead to new attacks (e.g. the attacks in [51, 50]).

3. *Full confusion and full diffusion are expected after modifying the round key of the first round.* As the secondary keys are unique and random, any difference in the first secondary key will be destroyed before using the second secondary key, the same goes for the decryption direction.

4. *The secondary keys should modify all the bits of the block.* Modifying a partial block increases the probability of certain attacks, like Square attack [36], where the adversary should find intermediate texts that vary in certain positions and are constant in the other positions.

Note that, if more than 2 round keys are needed to be modified, the above guidelines apply to the first and last rounds, where I[0]=x and I[m-1]=y.

Note in order for DPM to offer *differentiability*, the secondary key should be long enough to be resistant to *brute force attacks.*

4.9.3 Facts About AES

1. μ (defined in the above guidelines)=4 [143].

2. AES needs only 4 rounds to achieve full confusion and full diffusion properties [112].

4.9.4 Security of AESS1 and AESI1

For an adversary who does not know either the primary key or the secondary key, she tries to attack an AES with a random round key. AESS1 and AESI1 follow the guidelines in Sect. 4.9.1. After the modification, 5 to 7 rounds of AES are performed, these rounds assure that not only full confusion and diffusion after the modification are achieved, but also that any differential or linear propagation is completely destroyed. By using unique and random secondary keys the probability of finding secondary keys that share a certain pattern is considered very low, which limits the adversary's ability to mount a chosen plaintext attack. The same is valid for the decryption direction, as after the modification, 5 to 8 rounds of AES are performed, these rounds assure that not only full confusion and diffusion after the modification are achieved, but also that any differential or linear propagation is completely destroyed. By using unique and random secondary keys the probability of finding secondary keys that share a certain pattern is considered very low, which limits the adversary's ability to mount a chosen ciphertext attack. Thus, the adversary cannot mount linear, differential, chosen plaintext or chosen ciphertext attacks.

4.9.5 Security of AESS2 and AESI2

For an adversary who does not know either the primary key or the secondary key, she tries to attack a full round AES. AESS2 and AESI2 follow the guidelines in Sect. 4.9.2. After the first modification, 7 to 10 encryption rounds of AES are performed, these rounds assure that not only full confusion and diffusion after the modification are achieved, but also that any differential or linear propagation is completely destroyed in the encryption direction (as both the secondary keys are unique and random). After the second modification, 7 to 10 decryption rounds of AES are performed, these rounds assure that not only full confusion and diffusion after the modification are achieved, but also that any differential or linear propagation is completely destroyed in the decryption direction as both the secondary keys are unique and random. To the best of the author's knowledge, there is no feasible chosen plaintext attack that exists on 7 or 10 round AES, and by using random secondary keys the probability of finding secondary keys that share a certain pattern is considered very low. The same applies to chosen ciphertext attacks where there is no feasible ciphertext attack that exists on 7 or 10 round AES. Thus, the adversary cannot mount linear, differential, chosen plaintext or chosen ciphertext attacks.

4.9.6 Security of AES2S and AES2I

For an adversary who does not know either the primary key or the secondary key, she tries to attack a full round AES. AES2S and AES2I follow the guidelines in Sect. 4.9.2. After the first modification, 10 to 14 encryption rounds of AES are performed, these rounds assure that not only full confusion and diffusion after the modification are achieved, but also that any differential or linear propagation is completely destroyed in the encryption direction (as both the secondary keys

are unique and random). After the second modification, 7 to 14 decryption rounds of AES are performed, these rounds assure that not only full confusion and diffusion after the modification are achieved, but also that any differential or linear propagation is completely destroyed in the decryption direction as both the secondary keys are unique and random. To the best of the author's knowledge, there is no feasible chosen plaintext attack that exists on 10 or 14 round AES, and by using random secondary keys the probability of finding secondary keys that share a certain pattern is considered very low. The same applies to chosen ciphertext attacks where there is no feasible ciphertext attack that exists on 7, 10 or 14 round AES. Thus, the adversary cannot mount linear, differential, chosen plaintext or chosen ciphertext attacks.

Security of DP-AES

DP-AES offers variability and strong variability and follows most the guidelines in Sect. 4.9.1, but because AES-256 does not have many rounds, DP-AES cannot follow all the guidelines, thus the possibility of successful chosen plaintext/ciphertext attacks exists (with very low probability and under some extreme circumstances). Another issue is that the secondary key size is too small (24-bit), so it is not recommended using it to achieve differentiability. To conclude, it is not recommended using DP-AES as it is. In the next section, DP-AES will be used to construct a more secure variant of AES.

4.10 Hybrid Models

The proposed models can be used together to form new models. As an example, DPM and SSM are used to construct Dynamic Permutation Static Substitution AES (DPSS-AES).

4.10.1 Dynamic Permutation Static Substitution AES (DPSS-AES)

DPSS-AES uses both DPM and SSM. It accepts 280-bit secondary key. The listing of Encrypt-DPSS-AES, which is used for encryption in the proposed variant, can be found in table 4.11, where ***GetBits(SK,i,j)*** return from the i^{th} untill the j^{th} bits of **SK**. DPSS-AES works as follows:

- The 280-bit secondary key is divided into three parts; each part is responsible for a certain task:

  k1: is of size 24-bit and is used as the secondary key for Encrypt-DP-AES function (see table 4.10).

  k2 and k3: are of size 128-bit and are used to replace some words in AES' expanded key (k2 and k3 must be random and unique).

- **k2** replaces the round key of the 4^{th} round.
- **k2** replaces the round key of the 10^{th} round.
- The plaintext (**P**), updated expanded key (**EK**) and **k1** are passed to Encrypt-DP-AES function and the result is the ciphertext (**C**).

Table 4.11: Encrypt-DPSS-AES function.

```
Encrypt-DPSS-AES(P,EK,SK)
    k1=GetBits(SK,0,23)
    k2=GetBits(SK,24,151)
    k3=GetBits(SK,152,279)
Substitute( EK , k2 , 4 )
Substitute( EK , k3 , 10 )
C=Encrypt-DP-AES(P,EK,k1)
return C
```

Security of DPSS-AES

For an adversary who does not know either the primary key or the secondary key, she tries to attack a full round AES. DPSS-AES follows the guidelines in Sect. 4.9.2. After the first substitution, 10 encryption rounds of AES are performed, these rounds assure that not only full confusion and diffusion after the substitution are achieved, but also that any differential or linear propagation is completely destroyed in the encryption direction (as **k2** and **k3** are unique and random). After the second substitution, 10 decryption rounds of AES are performed, these rounds assure that not only full confusion and diffusion after the substitution are achieved, but also that any differential or linear propagation is completely destroyed in the decryption direction as both the secondary keys are unique and random). To the best of the author's knowledge, there is no feasible chosen plaintext attack that exists on 10 round AES, and by using random secondary keys the probability of finding secondary keys that share a certain pattern is considered very low. The same applies to chosen ciphertext attacks where there is no feasible ciphertext attack that exists on 10 round AES. Thus, the adversary cannot mount linear, differential, chosen plaintext or chosen ciphertext attacks.

4.10.2 Applications and Recommendations

The proposed models appeal for applications that require tweakable block ciphers, like disk encryption applications. Any variant of a block cipher constructed using these models, should be analyzed in the used scenario/application.

4.11 Summary

In this chapter, three new encryption models are proposed. These models allow any iterative block cipher to accept a secondary key; this was achieved by modifying the cipher's expanded key by substitution, permutation or addition. The guidelines to construct secure block ciphers using the proposed models are presented. The possibility to construct hybrid models by using these basic three models as building blocks is discussed. Based on the proposed models, several variants of AES that accept an extra secondary key are designed. These variants will be used in the next chapters to construct secure encryption schemes

for network applications and new modes of operation for disk encryption.

Chapter 5

Network Encryption Schemes

5.1 Introduction

The number of internet users is increasing continually world wide. Recent statistics reported that the current number of internet users is about 1.6 billion. This number has increased more than 350% in the last eight years and is still increasing daily [152]. Consequently, internet and network applications need to serve an increasing number of concurrent clients. At the same time, the enhanced quality and performance of internet and modern applications require more bandwidth capacity to fulfill the clients' needs. Today, modern networks do not only have to fulfill the demand of higher transmission rates but also have to provide and to guarantee data security and especially data confidentiality [84].

Key agility is particularly important in applications where only several blocks of data are encrypted between two consecutive key changes. IPSec [91, 92, 93] and ATM [41, 153, 69], with small packet sizes, and consecutive packets encrypted using different keys, are two widespread protocols in which the key setup latencies may play a very important role [63].

The two widely used schemes for generating the cipher's round keys in network applications are studied, where the Cipher Block Chaining mode (CBC) [117] and the Counter mode (CTR) [113] are used to perform encryption and decryption. The first scheme uses round keys pre-computation and the second uses on-the-fly round keys computation. Theoretical and experimental analyses are performed on both schemes. The analyses pointed out some shortcomings in both schemes, as the scheme that uses on-the-fly round keys generation performs more computations, on the other hand the scheme that uses round keys pre-computation uses more memory, which may limit the number of concurrent clients and is subjected to more cache misses and page faults causing instability in system performance.

To overcome the shortcomings of the current schemes, new schemes are proposed. The schemes are based on the Static Substitution Model (SSM) (refer to Sect. 4.5). AESS2 and AES2S (refer to Sect. 4.5.1) are used to construct encryption schemes for network applications. In the proposed schemes, each client possesses two keys. The first key is shared within a group of clients (a cluster) and it is expanded in memory (cluster key). The second key is the client's unique session key. Encryption and decryption are done using AESS2/AES2S,

where the cluster key is used to generate the expanded AES key and the client's session key is used as the secondary key. The proposed schemes enjoy high throughput together with low memory consumption.

The analysis is extended to the Galois/Counter Mode (GCM) [116], which has aroused to be one of the best methods for high speed authenticated encryption [116]. GCM is a block cipher mode of operation that uses universal hashing over a binary Galois field to provide authenticated encryption. GCM is built on the counter mode (CTR) [113] and it has been standardized by NIST [129]. There is a number of different software algorithms that implements universal hashing over a binary Galois field (GHASH), these implementations vary from their speed and memory requirements.

The different software implementations of GCM are studied. The analysis pointed out some shortcomings in each implementation, as the implementation that uses on-the-fly GHASH computation is considered slow. On the other hand the implementations that use pre-computed tables use more memory, which may limit the number of concurrent clients. GSCM is proposed to overcome these shortcomings.

5.2 Assumptions and Requirements

5.2.1 Assumptions

1. The server serves N concurrent secure sessions.
2. AES is used for encryption and decryption.
3. The encryption/decryption is done with the tested mode of operation.
4. Each client has two encryption keys, one for each traffic flow (the same method used in IPSec [91, 92, 93], when the Internet Key Exchange (IKE) [76] is used to establish fresh keys).
5. For decryption, the Equivalent Inverse Cipher of AES [37] is used, which has the same sequence of transformations as AES, thus offers a more efficient structure than the normal Inverse Cipher [127] and is used in many optimized software and hardware systems [66, 101, 102, 154] (Note that CTR and GCM schemes uses the same encryption algorithm in both encryption and decryption operations).

5.2.2 The Requirements of the Schemes

A high speed encryption scheme for networks needs to have:

1-High throughput: The faster the scheme the better.

2-Low memory: The less memory requirements the better.

3-Low CPU usage: The lower CPU usage the better.

4-Maximum number of clients: The more served clients the better.

5-Stability: The scheme should be stable, when the number of concurrent clients increases.

In order to examine the satisfaction of these requirements in current and proposed schemes, several simulation programs are developed. The design and the result of these simulations are discussed later in this chapter.

5.2.3 Secondary Keys Generation

In the proposed schemes, there are two classes of keys: the cluster keys and the clients' keys. It is proposed to use three keys for each cluster: Cluster Encryption Key **CEK**, Cluster Decryption Key **CDK** and Cluster User Key **CUK**. These three keys are unique for each cluster and are generated using a cryptographically secure random number generator. **CEK** and **CDK** are used to generate the cluster encryption and decryption expanded keys using the proposed enhanced AES key schedule (refer to Sect. 3.5). **CUK** is used to generate the clients' unique keys using the counter mode (CTR) [113]. In this way, each client has a unique pair of secondary keys.

5.3 Cipher Block Chaining (CBC) Schemes

In the following text, the standard widely used CBC schemes (CBC-Pre and CBC-On) and the proposed scheme CBC-S are presented.

5.3.1 CBC-Pre

CBC-Pre is a speed dedicated scheme, where the expanded key for encryption and decryption for each session are computed at the beginning of that session, stored in memory and then fetched from memory whenever an encryption or decryption operation for that session is needed. It executes as follows:

- **Setup Routine,** is executed for every client number i (C_i), once it is connected:
  - Two unique cryptographic random keys (K_i^1 and K_i^2) of size 128-bit or 256-bit are generated.
  - Two random initialization vectors (IV_i^1 and IV_i^2) of size 128-bit are generated.
  - K_i^1, K_i^2, IV_i^1 and IV_i^2 are sent to the client.
  - K_i^1 is expanded, using AES encryption key setup algorithm to produce the client's expanded encryption key (E_i).
  - K_i^2 is expanded, using AES decryption key setup algorithm to produce the client's expanded decryption key (D_i).
  - E_i, D_i, IV_i^1 and IV_i^2 are stored in server's memory.
- **Encryption Execution Routine,** to encrypt a plaintext (PT) for C_i:
  - E_i and IV_i^1 are fetched from the server's memory.
  - IV_i^1 is used to encrypt PT using CBC mode to produce CT, where E_i serves as the expanded encryption key.

  - CT and IV_i^1 are sent to the client (note that IV_i^1 is send to the client to ensure that the client can perform decryption, even when some packets are lost or reordered [80]).

- **Decryption Execution Routine,** to decrypt a ciphertext (CT) for C_i:
  - D_i is fetched from the server's memory.
  - IV_i^2 and CT are received from the client.
  - IV_i^2 is used to decrypt CT using CBC mode to produce PT, where D_i serves as the expanded decryption key.

5.3.2 CBC-On

CBC-On is a memory dedicated scheme, where the round keys for encryption and decryption are computed on-the-fly whenever an encryption or decryption operation is needed. It executes as follows:

- **Setup Routine,** is executed for every client number i (C_i), once it is connected:
  - Two unique cryptographic random keys (k_i^1 and k_i^2) of size 128-/256-bit are generated.
  - Two random initialization vectors (IV_i^1 and IV_i^2) of size 128-bit are generated.
  - k_i^1, k_i^2, IV_i^1 and IV_i^2 are sent to the client.
  - k_i^1, k_i^2, IV_i^1 and IV_i^2 are stored in the server's memory.

- **Encryption Execution Routine,** to encrypt a plaintext (PT) for C_i:
  - k_i^1 and IV_i^1 are fetched from the server's memory.
  - IV_i^1 is used to encrypt PT using CBC mode to produce CT, using k_i^1 as the encryption key, where the round keys are generated on-the-fly.
  - CT and IV_i^1 are sent to the client (note that IV_i^1 is send to the client to ensure that the client can perform decryption, even when some packets are lost or reordered [80]).

- **Decryption Execution Routine,** to decrypt a ciphertext (CT) for C_i:
  - k_i^2 is fetched from the server's memory.
  - IV_i^2 and CT are received from the client.
  - IV_i^2 is used to decrypt CT using CBC mode to produce PT, using k_i^2 as the encryption key, where the round keys are generated on-the-fly.

5.3.3 CBC-S

AESS2 (refer to Sect. 4.5.1) is used to build a scheme for network encryption applications, where:

- (K_{c1} and K_{c2}) are the cluster keys (for encryption and decryption respectively) and are shared by n clients (where n is the size of a cluster).

- Each client i (C_i) has its own two unique 128-bit keys (k_i^1) and (k_i^2).

CBC-S tries to eliminate the key setup latency, it executes as follows:

- **Cluster Setup Routine,** is used to setup a cluster of n-clients.
  - Three cryptographic secure random keys (K_{c1}, K_{c2} and K_u) with length 128-/256-bit are generated.
  - K_{c1} and K_{c2} are expanded, using the proposed enhanced AES key schedule (refer to Sect. 3.5) to produce the cluster's shared encryption/decryption expanded keys E_c and D_c.
  - E_c and D_c are stored in the server's memory.
- **Client Setup Routine,** is executed for every client number i (C_i), once it is connected:
  - Two unique cryptographic random keys (k_i^1 and k_i^2) of size 128-bit are generated, using K_u in the counter mode.
  - Two random initialization vectors (IV_i^1 and IV_i^2) of size 128-bit are generated.
  - K_{c1}, K_{c2}, k_i^1, k_i^2, IV_i^1 and IV_i^2 are sent to the client and stored in the server's memory. The client calculates E_c and D_c using K_{c1} and K_{c2}.
- **Encryption Execution Routine,** to encrypt a plaintext (PT) for C_i:
  - E_c, k_i^1, k_1^2 and IV_i^1 are fetched from the server's memory.
  - IV_i^1 is used to encrypt PT with CBC mode to produce CT using AESS2, where E_c serves as the expanded key, k_i^1 and k_i^2 construct the secondary key.
  - CT and IV_i^1 are sent to the client (note that IV_i^1 is sent to the client to ensure that the client can perform decryption, even when some packets are lost or reordered [80]).
- **Decryption Execution Routine,** to decrypt a ciphertext (CT) for C_i:
  - D_c, k_i^1, k_1^2 are fetched from the server's memory.
  - IV_i^2 and CT are received from the client.
  - IV_i^2 is used to decrypt CT with CBC mode to produce PT using AESS2, where D_c serves as the expanded key, k_i^1 and k_i^2 construct the secondary key.

5.4 Counter Mode (CTR) Schemes

5.4.1 Counter Block Format

Following the guidelines in [80], the counter block used in counter mode, has the following format:

1. The first 32-bit are a nonce, which are random and unique for each client.

2. The next 64-bit are the initialization vector (IV), which are random and incremented with each packet.
3. The last 32-bit are initialized for each packet by one, and incremented for each 128-bit block within the packet.

In the following text, the standard widely used CTR schemes (CTR-Pre and CTR-On) and the proposed scheme CTR-S are presented.

5.4.2 CTR-Pre

CTR-Pre is a speed dedicated scheme, where the expanded key for encryption and decryption for each session are computed at the beginning of that session, stored in memory and then fetched from memory whenever an encryption or decryption operation for that session is needed. It executes as follows:

- **Setup Routine,** is executed for every client number i (C_i), once it is connected:
  - Two unique cryptographic random keys (K_i^1 and K_i^2) of size 128-bit or 256-bit are generated.
  - Two random initialization vectors (IV_i^1 and IV_i^2) of size 64-bit are generated.
  - Two unique nonces (n_i^1 and n_i^2) of size 32-bit are generated.
  - K_i^1, K_i^2, IV_i^1, IV_i^2, n_i^1 and n_i^2 are sent to the client.
  - K_i^1 is expanded, using AES encryption key setup algorithm to produce the client's expanded encryption key (E_i).
  - K_i^2 is expanded, using AES decryption key setup algorithm to produce the client's expanded decryption key (D_i).
  - E_i, D_i, IV_i^1, IV_i^2, n_i^1 and n_i^2 are stored in server's memory.
- **Encryption Execution Routine,** to encrypt a plaintext (PT) for C_i:
  - E_i, IV_i^1 and n_i^1 are fetched from the server's memory.
  - IV_i^1 is incremented by one.
  - IV_i^1 and n_i^1 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to encrypt PT using CTR mode to produce CT, where E_i serves as the expanded encryption key.
  - CT and IV_i^1 are sent to the client (note that IV_i^1 is sent to the client to ensure that the client can generate the key stream needed for decryption, even when some packets are lost or reordered [80]).
- **Decryption Execution Routine,** to decrypt a ciphertext (CT) for C_i:
  - D_i and n_i^2 are fetched from the server's memory.
  - IV_i^2 and CT are received from the client.
  - IV_i^2 and n_i^2 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to decrypt CT using CTR mode to produce PT, where D_i serves as the expanded encryption key (note that the encryption function of the cipher is used in the decryption process of CTR mode).

5.4.3 CTR-On

CTR-On is a memory dedicated scheme, where the round keys for encryption and decryption are computed on-the-fly whenever an encryption or decryption operation is needed. It executes as follows:

- **Setup Routine,** is executed for every client number i (C_i), once it is connected:
  - Two unique cryptographic random keys (k_i^1 and k_i^2) of size 128-/256-bit are generated.
  - Two random initialization vectors (IV_i^1 and IV_i^2) of size 64-bit are generated.
  - Two unique nonces (n_i^1 and n_i^2) of size 32-bit are generated.
  - k_i^1, k_i^2, IV_i^1, IV_i^2, n_i^1 and n_i^2 are sent to the client and stored in the server's memory.
- **Encryption Execution Routine,** to encrypt a plaintext (PT) for C_i:
  - k_i^1, IV_i^1 and n_i^1 are fetched from the server's memory.
  - IV_i^1 is incremented by one.
  - IV_i^1 and n_i^1 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to encrypt PT using CTR mode to produce CT, using k_i^1 as the encryption key, where the round keys are generated on-the-fly.
  - CT and IV_i^1 are sent to the client (note that IV_i^1 is sent to the client to ensure that the client can generate the key stream needed for decryption, even when some packets are lost or reordered [80]).
- **Decryption Execution Routine,** to decrypt a ciphertext (CT) for C_i:
  - k_i^2 and n_i^2 are fetched from the server's memory.
  - IV_i^2 and CT are received from the client.
  - IV_i^2 and n_i^2 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to decrypt CT using CTR mode to produce PT, using k_i^2 as the encryption key, where the round keys are generated on-the-fly.

5.4.4 CTR-S

AES2S (refer to Sect. 4.5.1) is used to build a scheme for network encryption applications, where:

- (K_{c1} and K_{c2}) are the cluster keys (for encryption and decryption respectively) and are shared by n clients (where n is the size of a cluster).
- Each client i (C_i) has its own two unique 128-bit keys (k_i^1) and (k_i^2).

CTR-S tries to eliminate the key setup latency. It executes as follows:

- **Cluster Setup Routine,** is used to setup a cluster of n-clients.

  - Three cryptographic secure random keys (K_{c1}, K_{c2} and K_u) with length 128-/256-bit are generated.
  - K_{c1} and K_{c2} are expanded, using the proposed enhanced AES key schedule (refer to Sect. 3.5) to produce the cluster's shared encryption/decryption expanded keys (E_c) and (D_c).
  - E_c and D_c are stored in the server's memory.

- **Client Setup Routine,** is executed for every client number i (C_i), once it is connected:
  - Two unique cryptographic random keys (k_i^1 and k_i^2) of size 128-bit are generated, using K_u in the counter mode.
  - Two random initialization vectors (IV_i^1 and IV_i^2) of size 64-bit are generated.
  - Two unique nonces (n_i^1 and n_i^2) of size 32-bit are generated.
  - K_{c1}, K_{c2}, k_i^1, k_i^2, IV_i^1, IV_i^2, n_i^1 and n_i^2 are sent to the client. The client calculates E_c and D_c using K_{c1} and K_{c2}.
  - k_i^1, k_i^2, IV_i^1, IV_i^2, n_i^1 and n_i^2 are stored in the server's memory.

- **Encryption Execution Routine,** to encrypt a plaintext (PT) for C_i:
  - E_c, k_i^1, k_i^2, IV_i^1 and n_i^1 are fetched from the server's memory.
  - IV_i^1 is incremented by one.
  - IV_i^1 and n_i^1 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to encrypt PT with CTR mode to produce CT using AES2S, where E_c serves as the expanded key, k_i^1 and k_i^2 construct the secondary key.
  - CT and IV_i^1 are sent to the client (note that IV_i^1 is send to the client to ensure that the client can generate the key stream needed for decryption, even when some packets are lost or reordered [80]).

- **Decryption Execution Routine,** to decrypt a ciphertext (CT) for C_i:
  - D_c, k_i^1, k_i^2 and n_i^2 are fetched from the server's memory.
  - IV_i^2 and CT are received from the client.
  - IV_i^2 and n_i^2 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to decrypt CT with CTR mode to produce PT using AES2S, where D_c serves as the expanded key, k_i^1 and k_i^2 construct the secondary key (note that the encryption function of the cipher is used in the decryption process of CTR mode).

5.5 Galois/Counter Mode (GCM) Schemes

Galois/Counter Mode (GCM) [116] is a block cipher mode of operation that uses universal hashing over a binary Galois field to provide authenticated encryption. GCM is built on the counter mode (CTR) [113] and it has been standardized by NIST [129]. There is a number of different software algorithms that implements universal hashing over a binary Galois field (GHASH), these implementations vary from their speed and memory requirements.

5.5.1 GCM Software Implementations

GCM mode combines the well-known counter mode (CTR) [113] of encryption with the Galois mode of authentication. The key feature is that the Galois field multiplication used for authentication can be easily computed in parallel thus permitting higher throughput than the authentication algorithms that use chaining modes, like CBC [117].

GCM implementations with the different GHASH implementation strategies are tested. These implementations use on-the-fly strategy or pre-computed tables to compute GHASH. These pre-computed tables are of size: 256 bytes or 4Kb with Shoup's method [148], 8Kb in Gladman's implementation [66] and 64Kb with the straightforward method.

In the following text, the standard widely used GCM schemes (GCM(x)-Pre and GCM(x)-On) and the proposed scheme GSCM are presented.

5.5.2 GCM(x)-Pre

GCM(x)-Pre represents a group of five schemes, each scheme uses a different value of x. These values are 0, 256, 4096 (4k), 8192 (8k) and 65536 (64k) and they represent the bytes required for GHASH computation for each traffic flow per client. The pre-computed tables are computed at the beginning of the session, stored in memory and then fetched from memory whenever a GHASH computation is needed. It executes as follows:

- **Setup Routine,** is executed for every client number i (C_i), once it is connected:
  - Two unique cryptographic random keys (k_i^1 and k_i^2) of size 128-/256-bit are generated.
  - Two random initialization vectors (IV_i^1 and IV_i^2) of size 64-bit are generated.
  - Two unique nonces (n_i^1 and n_i^2) of size 32-bit are generated.
  - k_i^1, k_i^2, IV_i^1, IV_i^2, n_i^1 and n_i^2 are sent to the client.
  - k_i^1 and k_i^2 are expanded using AES encryption key setup algorithm to produce E_i and D_i.
  - T_i^1 and T_i^2 (tables of size x bytes used to compute the GHASH function in encryption and decryption directions respectively) are computed, using k_i^1 and k_i^2.
  - E_i, D_i, IV_i^1, IV_i^2, n_i^1, n_i^2, T_i^1 and T_i^2 are stored in the server's memory.

- **Encryption Execution Routine,** to encrypt a plaintext (PT) for C_i:
  - E_i, IV_i^1, n_i^1 and T_i^1 are fetched from the server's memory.
  - IV_i^1 is incremented by one.
  - IV_i^1 and n_i^1 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to encrypt PT using GCM mode to produce CT, where E_i serves as the expanded encryption key and T_i^1 is used to calculate the authentication tag t.
  - CT, IV_i^1 and t are sent to the client (note that IV_i^1 is sent to the client to ensure that the client can generate the key stream needed for decryption, even when some packets are lost or reordered [80]).
- **Decryption Execution Routine,** to decrypt a ciphertext (CT) for C_i:
  - D_i, n_i^2 and T_i^2 are fetched from the server's memory.
  - IV_i^2, CT and t are received from the client.
  - The authentication tag t' is calculated using T_i^2 and if t equals to t' the next step is executed, otherwise an authentication error is returned.
  - IV_i^2 and n_i^2 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to decrypt CT using GCM mode to produce PT, where D_i serves as the expanded encryption key (note that the encryption function of the cipher is used in the decryption process of CTR mode).

5.5.3 GCM(x)-On

GCM(x)-On represents a group of five schemes, each scheme uses a different value of x. These values are 0, 256, 4096 (4k), 8192 (8k) and 65536 (64k) and they represent the bytes required for GHASH computation for each traffic flow per client. The pre-computed tables are computed at the beginning of the session, stored in memory and then fetched whenever a GHASH computation is needed. It executes as follows:

- **Setup Routine,** is executed for every client number i (C_i), once it is connected:
  - Two unique cryptographic random keys (k_i^1 and k_i^2) of size 128-/256-bit are generated.
  - Two random initialization vectors (IV_i^1 and IV_i^2) of size 64-bit are generated.
  - Two unique nonces (n_i^1 and n_i^2) of size 32-bit are generated.
  - k_i^1, k_i^2, IV_i^1, IV_i^2, n_i^1 and n_i^2 are sent to the client.
  - T_i^1 and T_i^2 (tables of size x bytes used to compute the GHASH function in encryption and decryption directions respectively) are computed, using k_i^1 and k_i^2.
  - k_i^1, k_i^2, IV_i^1, IV_i^2, n_i^1, n_i^2, T_i^1 and T_i^2 are stored in server's memory.

- **Encryption Execution Routine,** to encrypt a plaintext (PT) for C_i:
  - k_i^1, IV_i^1, n_i^1 and T_i^1 are fetched from the server's memory.
  - IV_i^1 is incremented by one.
  - IV_i^1 and n_i^1 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to encrypt PT using GCM mode to produce CT, where k_i^1 serves as the encryption key (the round keys are calculated on-the-fly) and T_i^1 is used to calculate the authentication tag t.
  - CT, IV_i^1 and t are sent to the client (note that IV_i^1 is sent to the client to ensure that the client can generate the key stream needed for decryption, even when some packets are lost or reordered [80]).
- **Decryption Execution Routine,** to decrypt a ciphertext (CT) for C_i:
  - k_i^2, n_i^2 and T_i^2 are fetched from the server's memory.
  - IV_i^2, CT and t are received from the client.
  - The authentication tag t' is calculated using T_i^2 and if t equals to t' the next step is executed, otherwise an authentication error is returned.
  - IV_i^2 and n_i^2 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to decrypt CT using GCM mode to produce PT, where k_i^2 serves as the encryption key (the round keys are calculated on-the-fly), note that the encryption function of the cipher is used in the decryption process of CTR mode.

5.5.4 GSCM(x)

GSCM(x) represents a group of five schemes, each scheme uses a different value of x. These values are 0, 256, 4096 (4k), 8192 (8k) and 65536 (64k) and they represent the bytes required for GHASH computation for each traffic flow per client. AES2S (refer to Sect. 4.5.1) is used to build a scheme for network encryption applications, where:

- (K_{c1} and K_{c2}) are the cluster keys and are shared by n clients (where n is the maximum number of clients in a cluster).
- Each client i (C_i) has its own two unique 128-bit keys (k_i^1) and (k_i^2).

GSCM(x) tries to eliminate the setup latencies. It executes as follows:

- **Cluster setup routine:** is used to prepare the system and is executed for each cluster of n clients.
  - Three cryptographic secure random keys (K_{c1}, K_{c2} and K_u) with length 128-/256-bit are generated.
  - K_{c1} and K_{c2} are expanded using the proposed enhanced AES key schedule (refer to Sect. 3.5) to produce the cluster's shared encryption/decryption expanded keys (E_c) and (D_c).

  - T_{c1} and T_{c2} (tables of size x bytes used to compute the GHASH function in encryption and decryption directions respectively) are computed, using K_{c1} and K_{c2}.
  - E_c, D_c, T_{c1} and T_{c2} are stored in the server's memory.

- **Client setup routine:** is executed for every client number i (C_i), once it is connected:
  - Two unique cryptographic random keys (k_i^1 and k_i^2) of size 128-bit are generated, using K_u in the counter mode.
  - Two random initialization vectors (IV_i^1 and IV_i^2) of size 64-bit are generated.
  - Two unique nonces (n_i^1 and n_i^2) of size 32-bit are generated.
  - K_{c1}, K_{c2}, k_i^1, k_i^2, K_{c1}, K_{c2}, IV_i^1, IV_i^2, n_i^1 and n_i^2 are sent to the client. The client calculates E_c, D_c, T_{c1} and T_{c2} using K_{c1} and K_{c2}.
  - k_i^1, k_i^2, IV_i^1, IV_i^2, n_i^1 and n_i^2 are stored in the server's memory.

- **Encryption execution routine,** to encrypt a plaintext (PT) for C_i:
  - E_c,k_i^1, k_i^2, IV_i^1, n_i^1 and T_{c1} are fetched from the server's memory.
  - IV_i^1 is incremented by one.
  - IV_i^1 and n_i^1 are used to construct the initial counter block (ICB_i) for GCM mode (as explained in Sect. 5.4.1).
  - ICB_i is used to encrypt PT with GCM mode to produce CT using AES2S, where E_c serves as the expanded key, k_i^1 and k_i^2 construct the secondary key. T_{c1} is used to calculate the authentication tag t.
  - CT, IV_i^1 and t are sent to the client (note that IV_i^1 is sent to the client to ensure that the client can generate the key stream needed for decryption, even when some packets are lost or reordered [80]).

- **Decryption execution routine,** to decrypt a ciphertext (CT) for C_i:
  - D_c,k_i^1, k_i^2, n_i^2 and T_{c2} are fetched from the server's memory.
  - IV_i^2, CT and t are received from the client.
  - The authentication tag t' is calculated using T_{c2} and if t equals to t' the next step is executed, otherwise an authentication error is returned.
  - IV_i^2 and n_i^2 are used to construct the initial counter block (ICB_i) for CTR mode (as explained in Sect. 5.4.1).
  - ICB_i is used to decrypt CT with GCM mode to produce PT using AES2S, where D_c serves as the expanded key, k_i^1 and k_i^2 construct the secondary key (note that the encryption function of the cipher is used in the decryption process of CTR mode).

5.6 Memory Analysis

The less memory the scheme needs, the more available memory to other applications and the larger the number of concurrent clients the server can serve. Memory access has a great role in the overall scheme performance. If a server has insufficient physical memory space to cache all of the data it needs, it performs page replacements. Although virtual memory makes the server able to complete the process, this process is very slow compared to the cost of data computation [103].

Table 5.1 represents the memory requirements in bytes for CBC and CTR schemes per client to hold the key material and other data required to perform encryption and decryption (e.g. initialization vectors, nonces, counter, etc..). Note that the memory requirements for CBC and CTR modes are the same for each scheme. From these results, it is clear that the proposed schemes CBC-S and CTR-S possess the lowest memory requirements and thus can serve the maximum number of concurrent clients. When 128-bit keys are used, the proposed schemes can serve more than 12 times the number of clients sereved by CBC-Pre/CTR-Pre and more than 3 times the number of clients sereved by CBC-On/CTR-On, moreover when 256-bit keys are used these factors are increased to 16 and 5 respectively.

Table 5.1: Memory requirements for CBC and CTR schemes per client (in bytes).

	128-bit key	256-bit key
CTR-Pre/CBC-Pre	784	1040
CTR-On/CBC-On	208	336
CTR-S/CBC-S	64	64

Table 5.2 represents the memory requirements in bytes for GCM schemes per client to hold the key material and other data required to perform encryption and decryption (e.g. initialization vectors, nonces, counter, etc..). From these results, it is clear that GSCM(x) schemes possess the lowest memory requirements and thus can serve the maximum number of concurrent clients. GSCM(x) schemes can serve at least twice the number of clients than the current schemes and in some cases this factor increases to about 1029.

5.7 Server Configuration and Simulation Parameters

5.7.1 Server Configuration

NoCrypto and the other tested schemes are implemented using C++ language and several simulations are performed to examine their practical behavior. Table 5.3 shows the server configuration used in the simulations. Note that NoCrypto does not perform any encryption or decryption operations, it is presented here to show the cryptographic overhead and in the proposed schemes all the clients share the same cluster.

Table 5.2: Memory requirements for GCM schemes per client (in bytes).

	128-bit key	256-bit key
GCM(64k)-Pre	131624	131752
GCM(64k)-On	131336	131400
GCM(8k)-Pre	16936	17064
GCM(8k)-On	16648	16712
GCM(4k)-Pre	8744	8872
GCM(4k)-On	8456	8520
GCM(256)-Pre	1064	1192
GCM(256)-On	776	840
GCM(0)-Pre	552	680
GCM(0)-On	264	328
GSCM(x)	128	128

Table 5.3: Server configuration.

Processor	Intel Xeon Quad-Core 2.33 GHz (64-bit)
RAM	4096 MB
Processor Cache	12 MB
Paging file	4096 MB
OS	Microsoft Windows Server 2008
Compiler	Visual C++ 2008
Code optimization	Maximum speed

5.7.2 Parameters

For the simulation instances, the following parameters are used:

- α is the tested packet size; it is either 40 or 1500 bytes. This is because the distribution of Internet packet sizes seems mostly bimodal at 40 and 1500 bytes [68, 149].
- **UNIT** is the current number of clients.
- **STEP** is a number used to calculate the current number of clients in a given simulation instance.
- $\mathbf{Z}_i$ is the current number of clients served by the server, the set Z = { Z_i } is constructed, where Z_i = (i × STEP).
- **K** is the key size in bits (either 128 or 256).

5.8 Stability Analysis

In order to gain more insight into the practical behavior of the schemes, a simulation program is designed to explore how the schemes behave when the number of clients increases.

5.8.1 The Stability Analysis Simulation

A multi-client/server TCP socket application is developed to demonstrate the behavior of the schemes. The server and the clients are connected via a LAN (100 Mbps). The simulation works as follows:

1. The server allocates 90% of its available RAM as a shared data pool. This pool holds the encrypted and decrypted data for all the clients and is used to illustrate the effect of reading and writing to the RAM.
2. The server allocates the memory needed by the tested scheme to serve Z_i clients, where for each client the server allocates M bytes, where M is the number of bytes required by each client using the tested scheme (refer to table 5.1 and table 5.2).
3. The server waits until 50 computers are connected, then sends the start command to all the computers (each computer simulates $Z_i/50$ clients). Note that each computer is served using a different thread, to illustrate the effect of multi-threading.
4. As the computer receives the start command, it sends a packet of size α to the server.
5. The packet is decrypted for client C_i and encrypted to client C_x, where i and x are positive random numbers less than Z_i.
6. The server sends the encrypted packet (from the previous step), to the computer that serves client C_x.

7. When the packet is processed by the server and received by the computer, the computer starts to send a packet for the next client it simulates.

8. The average time (in milliseconds) for processing a packet is reported.

9. Note that, it is assumed that the incoming and outgoing packets are of the same size for simplicity.

The simulation is performed, where STEP=100,000 with the following termination conditions:

1. When the average packet processing time for a packet exceeds that of NoCrypto with a factor of 2 (using the same parameters).

2. The server cannot allocate memory for Z_i clients.

3. The server begins to lose its connections.

5.8.2 The Results of the Stability Analysis

Figure 5.1 and Figure 5.2 show the simulation results of CBC schemes. From these results it is clear that the encryption schemes reach instability after a certain threshold; this instability is due to cache misses and page faults.

In table 5.4, the maximum number of clients is reported in the simulation, together with the maximum number of stable clients. These numbers are the average of the recored values for α=40 and α=1500. If K equals 128-bit, the simulation reported that CBC-S can serve about 635% and 160% stable clients more than CBC-Pre and CBC-On respectively; when K equals 256-bit these percentages are increased to 690% and 265%.

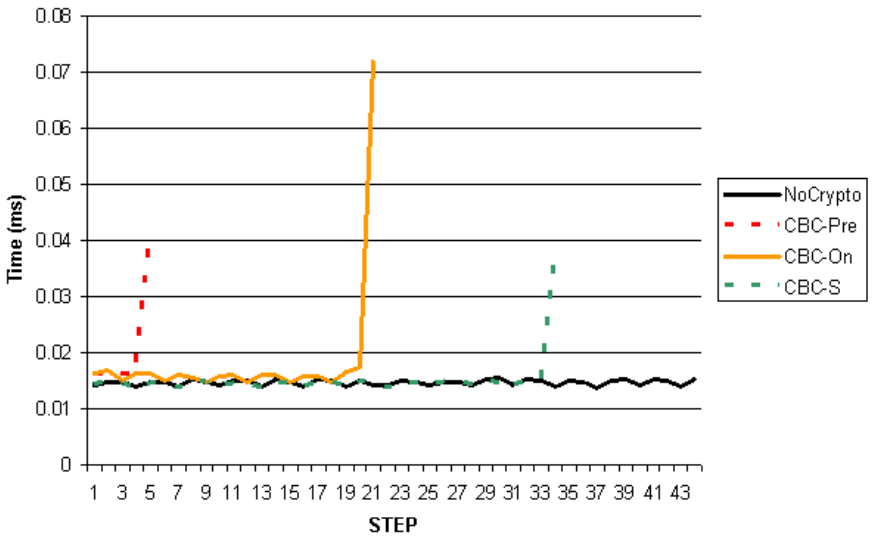
Table 5.4: Maximum reported number of clients and stable number of clients for CBC schemes.

	Maximum Reported Number of Clients		Maximum Reported Stable Number of Clients	
Key Size	128-bit	256-bit	128-bit	256-bit
CBC-Pre	700000	600000	550000	500000
CBC-On	2400000	1550000	2200000	1300000
CBC-S	3750000	3750000	3500000	3450000

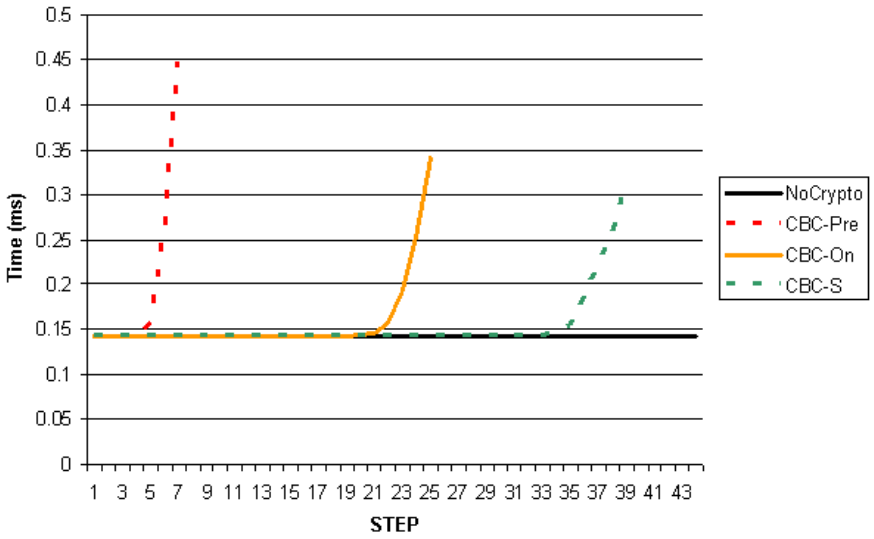
Figure 5.3 and Figure 5.4 show the simulation results of CTR schemes. From these results it is clear that the encryption schemes reach instability after a certain threshold; this instability is due to cache misses and page faults.

In table 5.5, the maximum number of clients is reported in the simulation, together with the maximum number of stable clients. These numbers are the average of the recored values for α=40 and α=1500. If K equals 128-bit, the simulation reported that CTC-S can serve about 645% and 170% stable clients as CTR-Pre and CTR-On respectively; when K equals 256-bit these percentages increase to 690% and 255%.

In table 5.6, the maximum number of clients is reported in the simulation, together with the maximum number of stable clients reported for all GCM

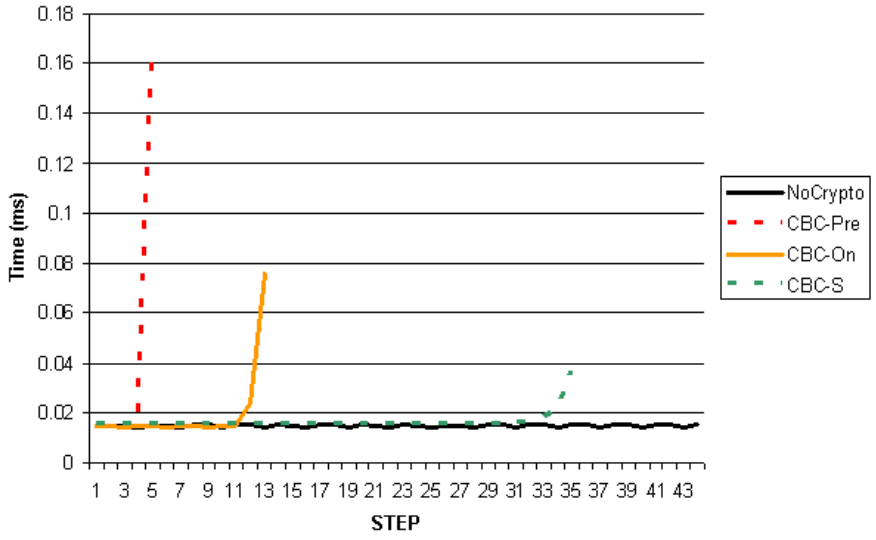


(a) Simulation parameters (α=40 and STEP=100,000).

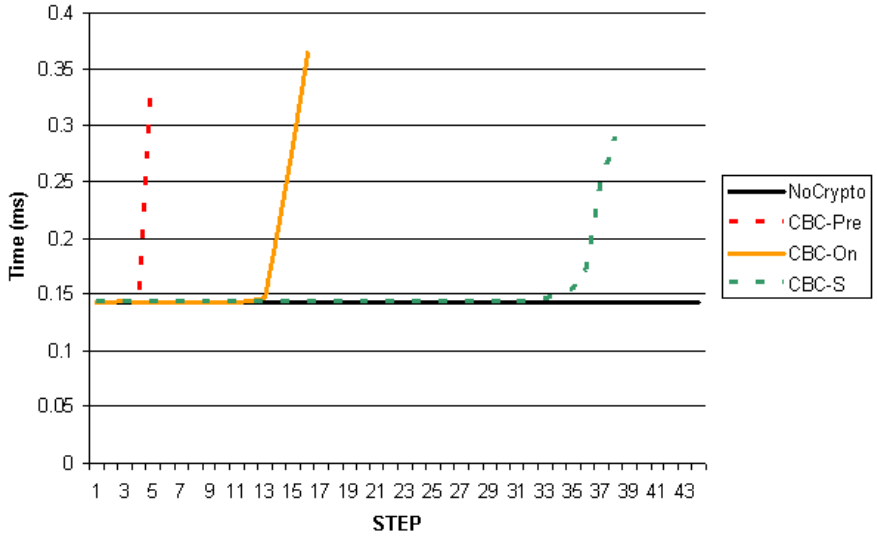


(b) Simulation parameters (α=1500 and STEP=100,000).

Figure 5.1: Stability analysis simulation: average time (in ms) needed to process a packet, using CBC schemes (K=128).

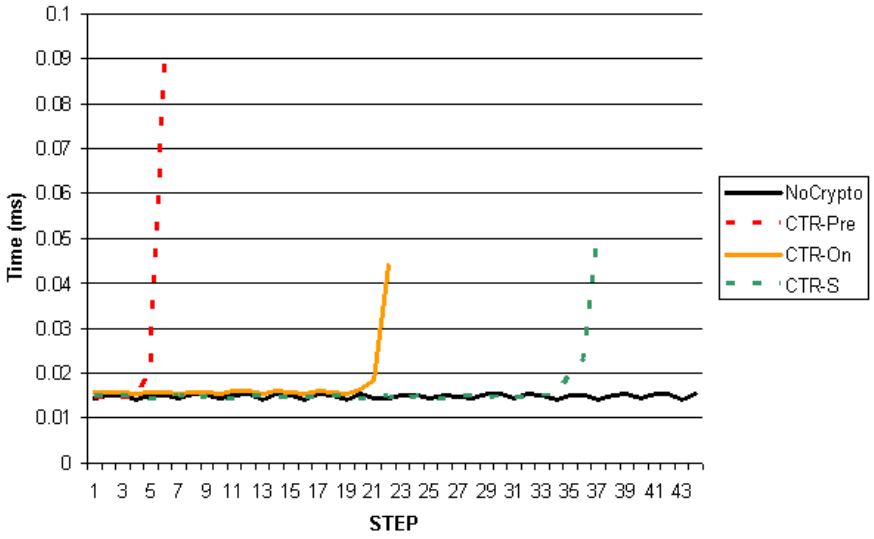


(a) Simulation parameters (α=40 and STEP=100,000).

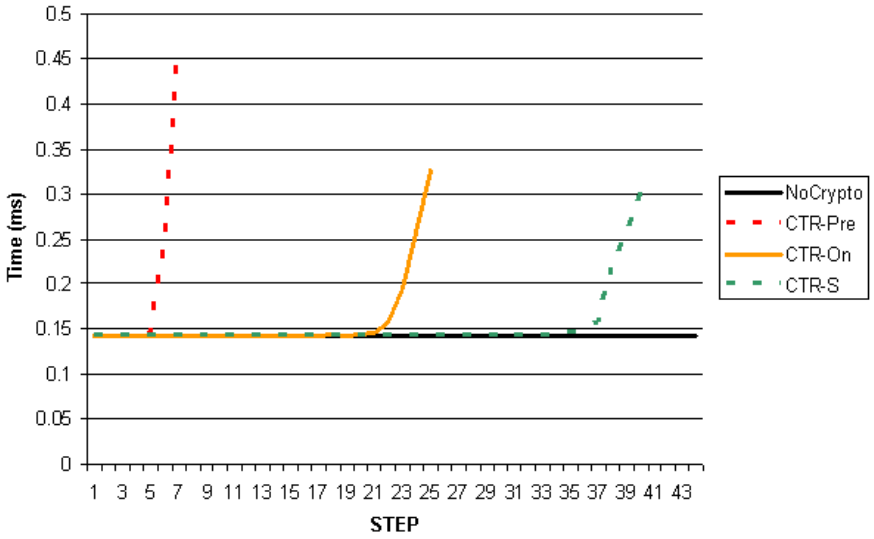


(b) Simulation parameters (α=1500 and STEP=100,000).

Figure 5.2: Stability analysis simulation: average time (in ms) needed to process a packet, using CBC schemes (K=256).

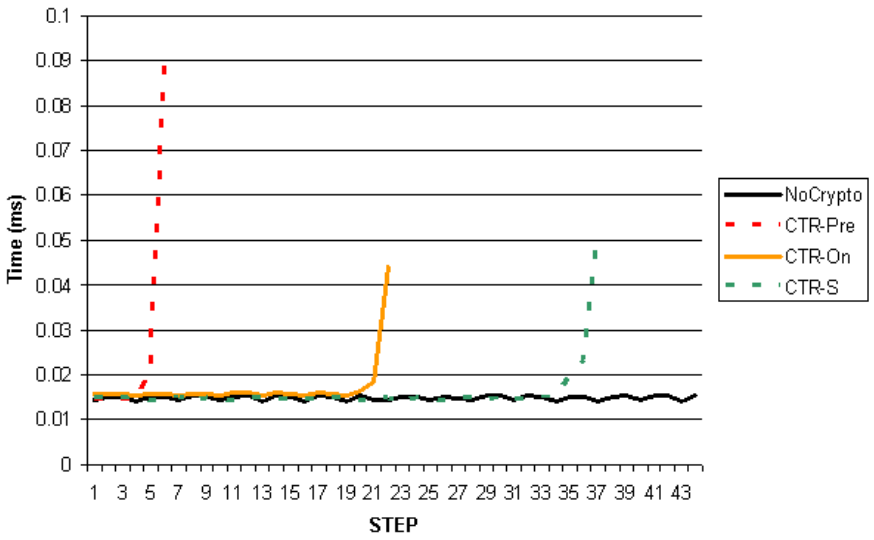


(a) Simulation parameters (α=40 and STEP=10,000).

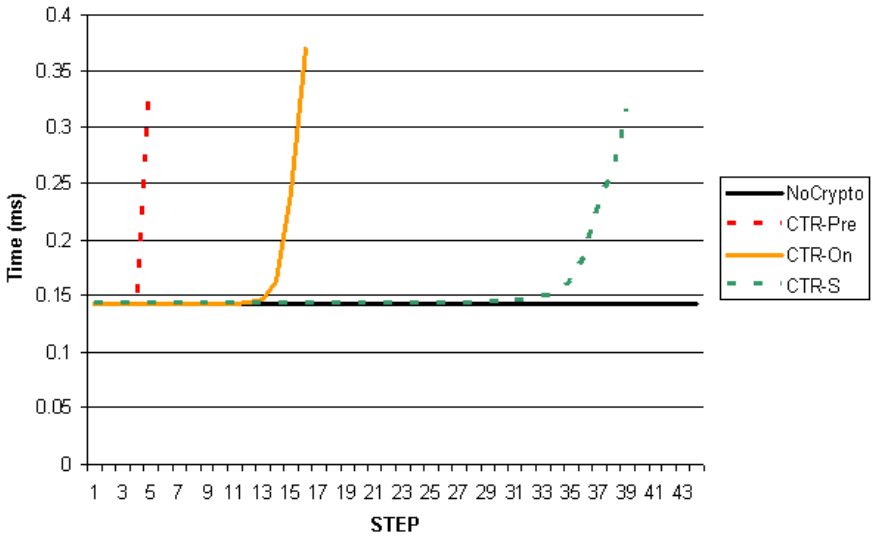


(b) Simulation parameters (α=1500 and STEP=10,000).

Figure 5.3: Stability analysis simulation: average time (in ms) needed to process a packet, using CTR schemes (K=128).



(a) Simulation parameters (α=40 and STEP=10,000).



(b) Simulation parameters (α=1500 and STEP=10,000).

Figure 5.4: Stability analysis simulation: average time (in ms) needed to process a packet, using CTR schemes (K=256).

Table 5.5: Maximum reported number of clients and stable number of clients for CTR schemes.

	Maximum Reported Number of Clients		Maximum Reported Stable Number of Clients	
Key Size	128-bit	256-bit	128-bit	256-bit
CTR-Pre	750000	600000	550000	500000
CTR-On	2450000	1550000	2100000	1350000
CTR-S	3950000	3850000	3550000	3450000

schemes. These numbers are the average of the recored values for α=40 and α=1500.

It is worth mentioning that the maximum number of stable clients with α=1500 is greater than that with α=40 for all the schemes.

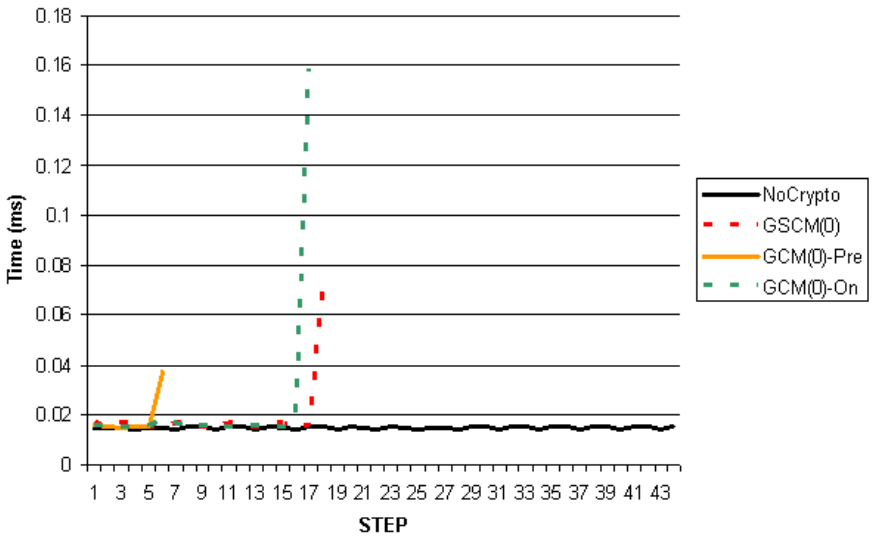
Figure 5.5 and Figure 5.6 show the simulation results for GCM(0) schemes. From these results it is clear that the encryption schemes reach instability after a certain threshold; this instability is due to cache misses and page faults. If K equals 128-bit, the simulation reported that GSCM(0) can serve about 265% and 105% stable clients as GCM(0)-Pre and GCM(0)-On respectively; when K equals 256-bit these percentages increase to 320% and 130%.

Figure 5.7 and Figure 5.8 show the simulation results for GCM(256) schemes. From these results it is clear that the encryption schemes reach instability after a certain threshold; this instability is due to cache misses and page faults. If K equals 128-bit, the simulation reported that GSCM(256) can serve about 530% and 410% stable clients as GCM(256)-Pre and GCM(256)-On respectively; when K equals 256-bit these percentages increase to 600% and 450%.

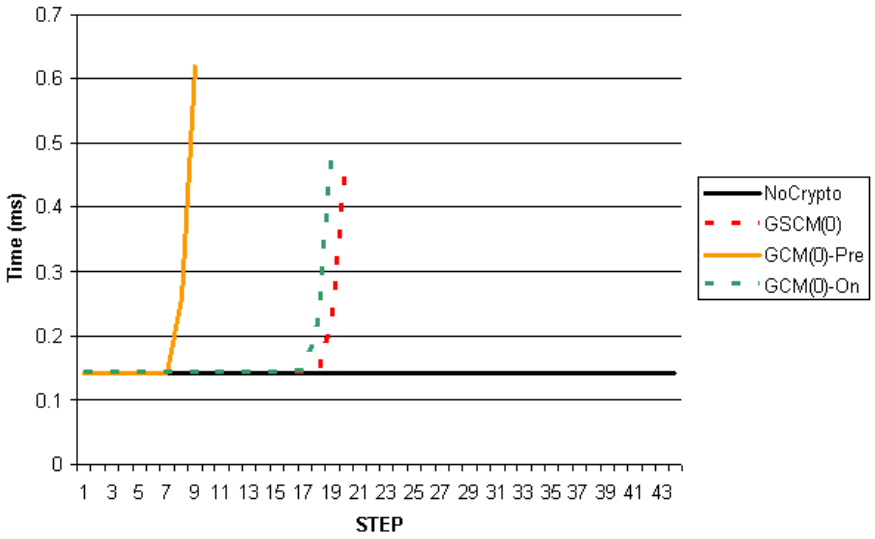
Table 5.6: Maximum reported number of clients and stable number of clients for GCM schemes.

	Maximum Reported Number of Clients		Maximum Reported Stable Number of Clients	
Key Size	128-bit	256-bit	128-bit	256-bit
GCM(0)-Pre	850000	700000	700000	550000
GCM(0)-On	1900000	1500000	1750000	1350000
GCM(256)-Pre	450000	450000	350000	300000
GCM(256)-On	600000	500000	450000	400000
GCM(4k)-Pre	200000	200000	50000	50000
GCM(4k)-On	200000	200000	50000	50000
GCM(8k)-Pre	200000	200000	50000	50000
GCM(8k)-On	200000	200000	50000	50000
GCM(64k)-Pre	NEM	NEM	NEM	NEM
GCM(64k)-On	NEM	NEM	NEM	NEM
GSCM(x)	2000000	1950000	1850000	1800000
NEM = Not Enough Memory				

For both 128-bit keys and 256-bit keys, the simulation reported that GSCM(4k) can serve about 3700% and 3600% stable clients as GCM(4k)-Pre

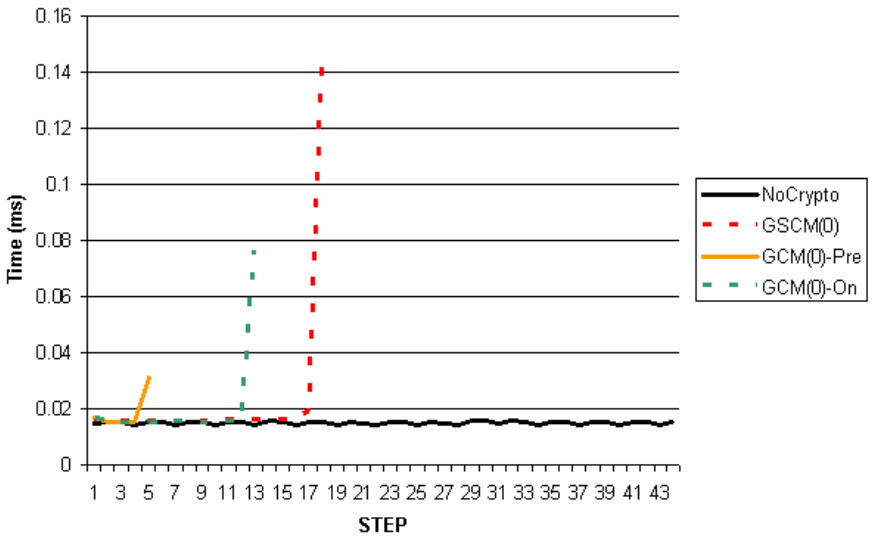


(a) Simulation parameters (α=40 and STEP=10,000).

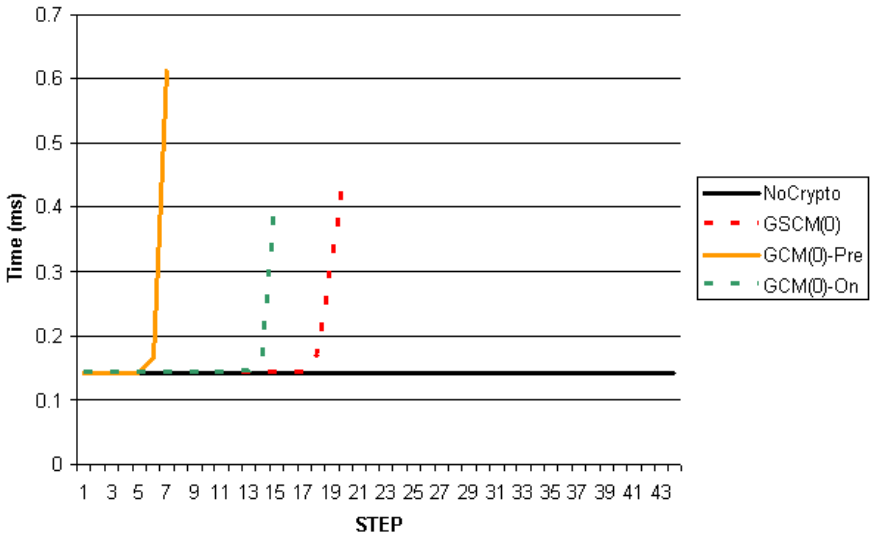


(b) Simulation parameters (α=1500 and STEP=10,000).

Figure 5.5: Stability analysis simulation: average time (in ms) needed to process a packet, using GCM(0) Schemes (K=128).

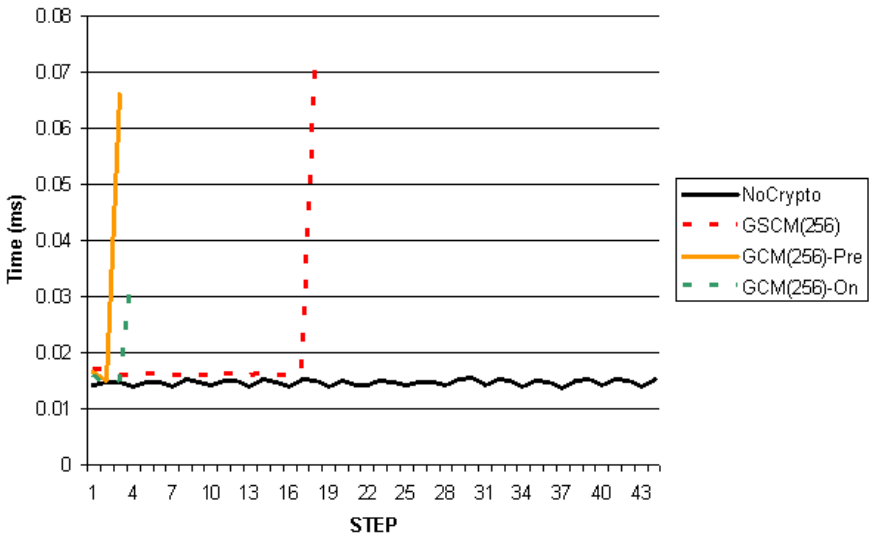


(a) Simulation parameters (α=40 and STEP=10,000).

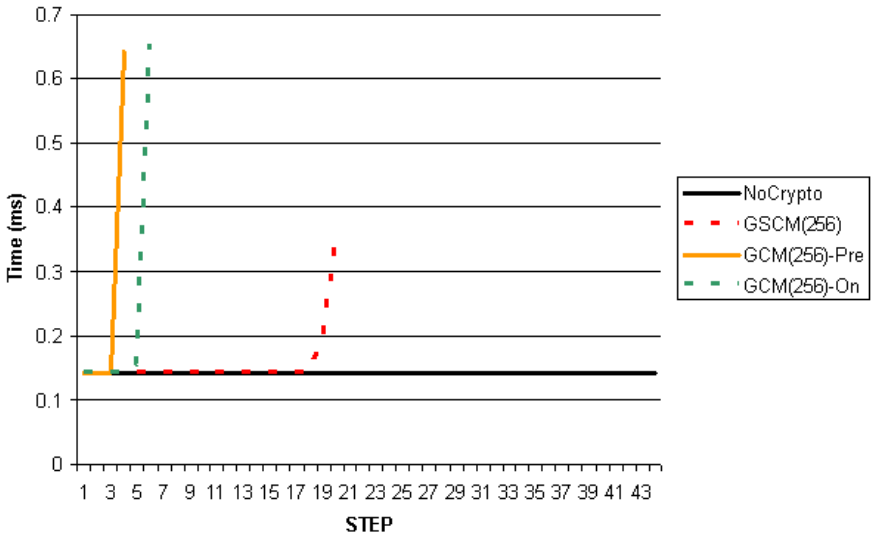


(b) Simulation parameters (α=1500 and STEP=10,000).

Figure 5.6: Stability analysis simulation: average time (in ms) needed to process a packet, using GCM(0) Schemes (K=256).

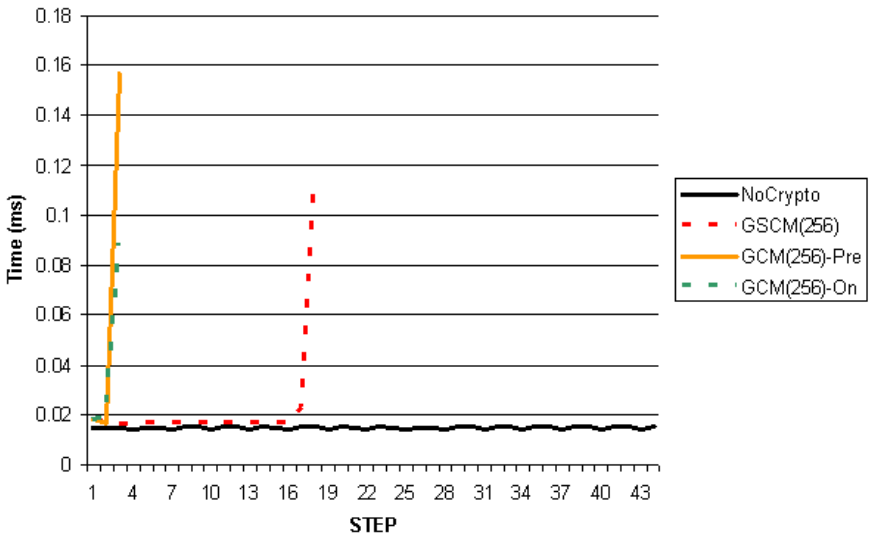


(a) Simulation parameters (α=40 and STEP=10,000).

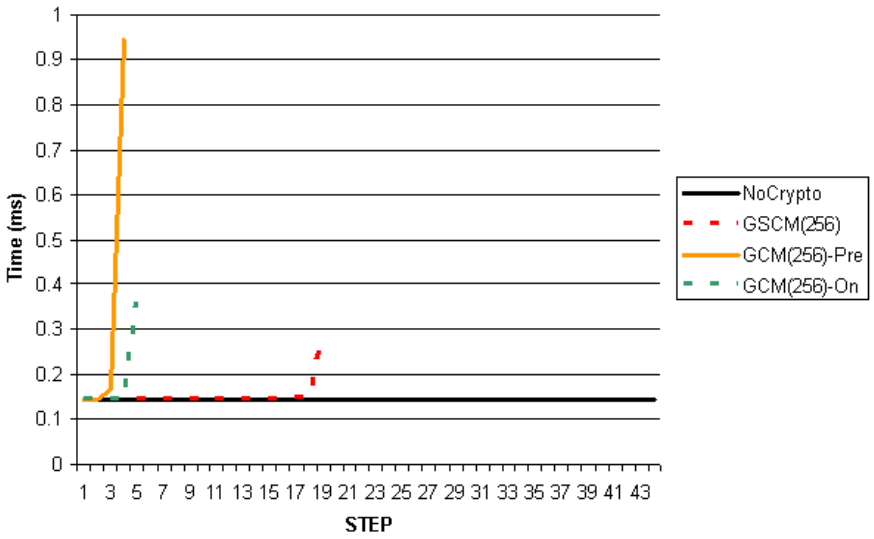


(b) Simulation parameters (α=1500 and STEP=10,000).

Figure 5.7: Stability analysis simulation: average time (in ms) needed to process a packet, using GCM(256) Schemes (K=128).



(a) Simulation parameters (α=40 and STEP=10,000).



(b) Simulation parameters (α=1500 and STEP=10,000).

Figure 5.8: Stability analysis simulation: average time (in ms) needed to process a packet, using GCM(256) Schemes (K=256).

and GCM(4k)-On respectively. For both 128-bit keys and 256-bit keys, the simulation reported that GSCM(8k) can serve about 3700% and 3600% stable clients as GCM(8k)-Pre and GCM(8k)-On respectively.

In case of GCM(64k)-Pre and GCM(64k)-On the simulation program failed to allocate the required memory, due to the high memory requirements for these schemes.

5.9 Performance Analysis

5.9.1 The Performance Analysis Simulation

A multi-threaded application is developed to demonstrate the performance of the schemes. The simulation works as follows:

1. The server allocates 90% of its available RAM as a shared data pool. This pool holds the encrypted and decrypted data for all the clients and is used to illustrate the effect of reading and writing to the RAM.

2. The server allocates the memory needed by the tested scheme to serve UNIT clients, where for each client, the server allocates M bytes, where M is the number of bytes required by each client using the tested scheme (refer to table 5.1 and table 5.2).

3. The server starts 50 threads to illustrate the effect of multi-threading.

4. Each thread simulates UNIT/50 clients.

5. For each client 10 random packets are decrypted for client C_i and encrypted to client C_x, where i and x are positive random numbers less than UNIT.

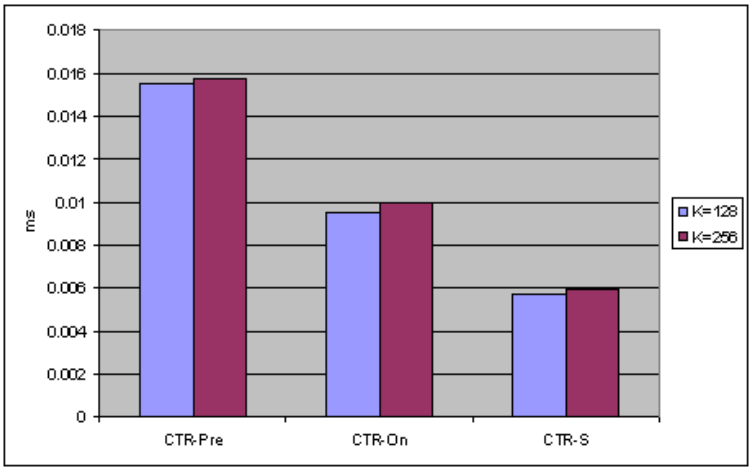
6. The average time (in milliseconds) for processing a packet is reported.

The simulation is performed, where UNIT=100,000 for CBC and CTR schemes and UNIT=10,000 for GCM schemes.

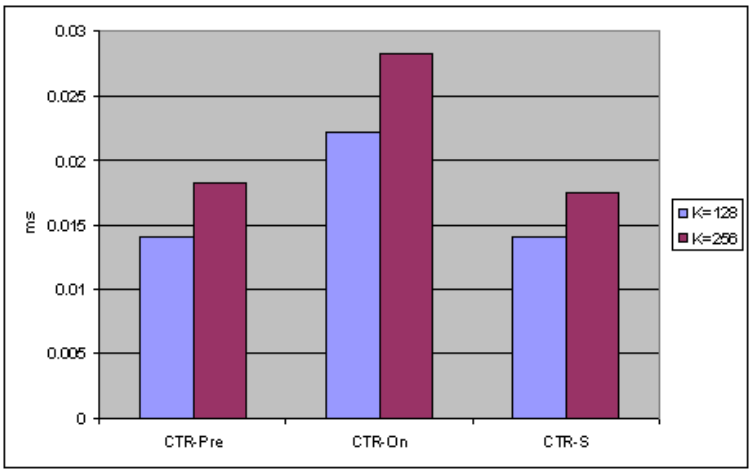
5.9.2 The Results of the Performance Analysis

The results of the performance simulation for CBC schemes are shown in Figure 5.9. These results demonstrate that CBC-S is faster than CBC-Pre and CBC-On. For small packets ($\alpha = 40$), CBC-S is about 165 to 175% faster than CBC-Pre and 65 to 67% faster than CBC-On (depending on the key size). Moreover, these results also demonstrate that for large packets, CBC-S is faster than CBC-Pre by about 5% (when K=256-bit), possess the same speed as CBC-Pre (when K=128-bit) and faster than CBC-On by about 55 to 60%.

The results of the performance simulation for CTR schemes are shown in Figure 5.10. These results demonstrate that CTR-S is faster than CTR-Pre and CTR-On when $\alpha = 40$, where it is about 95 to 135% faster than CBC-Pre and 30 to 40% faster than CTR-On (depending on the key size). Moreover, these results also demonstrate that for large packets, CTR-S is faster than CTR-Pre by about 2 to 10% and faster than CTR-On by about 20 to 45%.

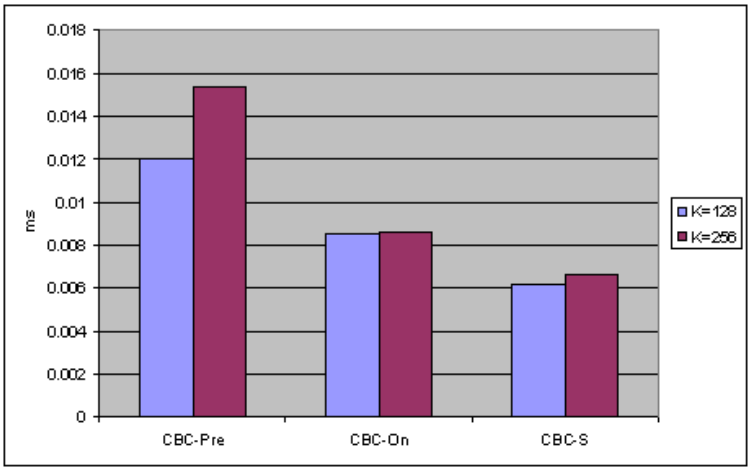


(a) Simulation parameters (α=40 and UNIT=100,000).

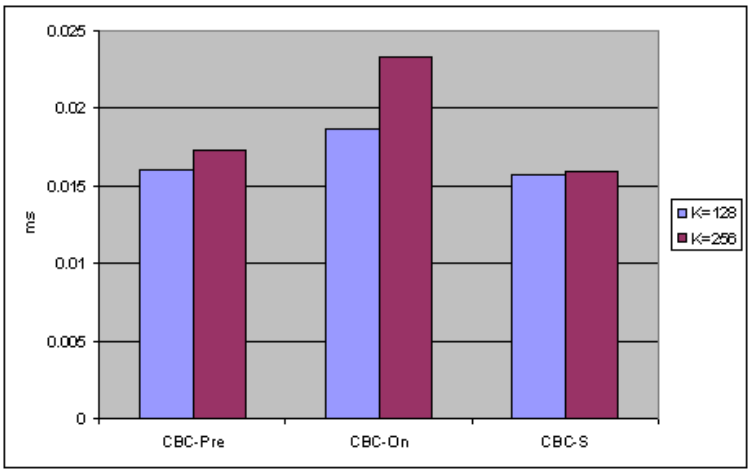


(b) Simulation parameters (α=1500 and UNIT=100,000).

Figure 5.9: Performance analysis simulation: average time (in ms) needed to process a packet, using CBC schemes.



(a) Simulation parameters (α=40 and UNIT=100,000).



(b) Simulation parameters (α=1500 and UNIT=100,000).

Figure 5.10: Performance analysis simulation: average time (in ms) needed to process a packet, using CTR schemes.

Table 5.7 and table 5.8 present the Speed up percentages of GSCM(x) schemes over GCM(x)-Pre and GCM(x)-On schemes. The speed up percentage is calculated by (5.1). For more details on the simulation results refer to Appendix B.

$$SpeedUp = (\frac{\text{GSCM Speed}}{\text{GCM Speed}} - 1) \times 100 \tag{5.1}$$

Table 5.7: Speed up percentages of GSCM(x) schemes over GCM(x)-Pre and GCM(x)-On schemes ($\alpha = 40$).

x	GCM(x)-Pre 128-bit	GCM(x)-Pre 256-bit	GCM(x)-On 128-bit	GCM(x)-On 256-bit
0	23	8	18	8
256	20	21	27	22
4k	30	13	45	25
8k	44	44	53	44
64k	270	215	280	215

Table 5.8: Speed up percentages of GSCM(x) schemes over GCM(x)-Pre and GCM(x)-On schemes ($\alpha = 1500$).

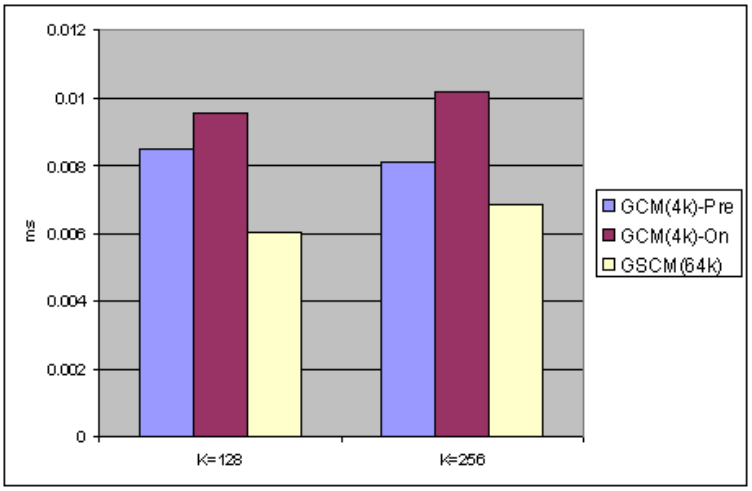
x	GCM(x)-Pre 128-bit	GCM(x)-Pre 256-bit	GCM(x)-On 128-bit	GCM(x)-On 256-bit
0	-3	-2	2	8
256	0	0	9	13
4k	2	0	15	20
8k	3	0	13	18
64k	55	60	70	85

Figure 5.11, presents the simulation results for the fastest GCM schemes. GSCM(64k) is considered the fastest GSCM(x), GCM(4k)-Pre is the fastest GCM(x)-Pre and GCM(4k)-On is the fastest GCM(x)-On. GSCM(64k) is faster than GCM(4k)-Pre and GCM(4k)-On when $\alpha = 40$, where it is about 19 to 40% faster than GCM(4k)-Pre and 49% to 58% faster than GCM(4k)-On. Moreover, GSCM(64k) is faster than GCM(4k)-Pre by about 7 to 8% and it is faster than GCM(4k)-On by about 20% to 30% (when α=1500).

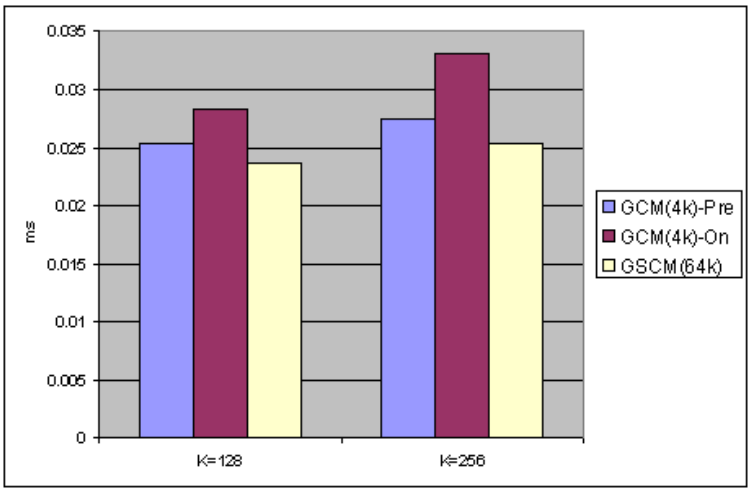
5.10 Network Analysis

5.10.1 The Network Analysis Simulation

A multi-client/server TCP socket application is developed to demonstrate the behavior of the schemes. The server and the clients are connected via a LAN (100 Mbps). The scenarios work as follows:



(a) (α=40 and STEP=10,000).



(b) (α=1500 and STEP=10,000).

Figure 5.11: Performance analysis simulation: average time (in ms) needed to process a packet by the fastest GCM schemes.

1. The server allocates 50% of its available RAM as a shared data pool. This pool holds the encrypted and decrypted data for all the clients and is used to illustrate the effect of reading and writing to the RAM.

2. The server allocates the memory needed by the tested scheme to serve UNIT clients, where for each client, the server allocates M bytes, where M is the number of bytes required by each client using the tested scheme (refer to table 5.1 and table 5.2).

3. The server waits until 50 computers are connected, then sends the start command to all the computers (each computer simulates UNIT/50 clients). Note that each computer is served using a different thread, to illustrate the effect of multi-threading.

4. As the computer receives the start command, it sends a packet of size α to the server.

5. The packet is decrypted for client C_i and encrypted to client C_x, where i and x are positive random numbers less than UNIT.

6. The server sends the encrypted packet (from the previous step), to the computer that serves client C_x.

7. When the packet is processed by the server and received by the computer, the computer starts to send another 9 packets (one at a time), after that it starts to send packets for the next client it simulates.

8. The average time (in milliseconds) for processing a packet is reported.

5.10.2 The Results of the Network Analysis

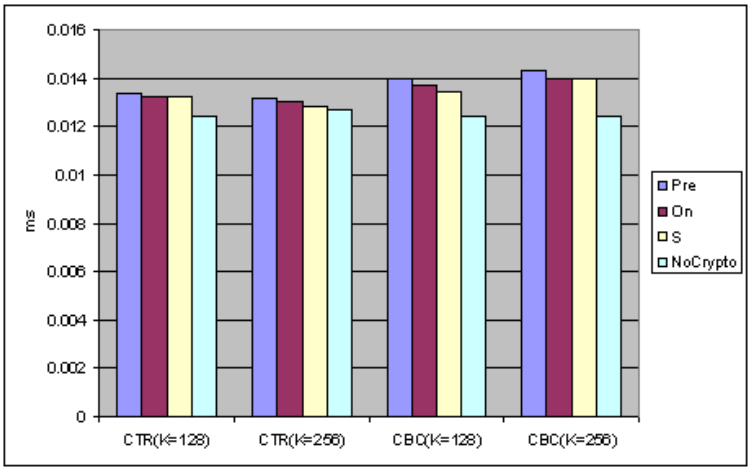
Note that UNIT is chosen to be 100,000 to get a fair analysis, because if UNIT increases some schemes may be instable and the average time to process a packet will be high. The amount occupied by data pool is also decreased to 50% of the server's memory, for GCM schemes UNIT is reduced to 10,000. The Overhead percentage is calculated by (5.2).

$$Overhead = (\frac{\text{Scheme Speed}}{\text{NoCrypto Speed}} - 1) \times 100 \tag{5.2}$$

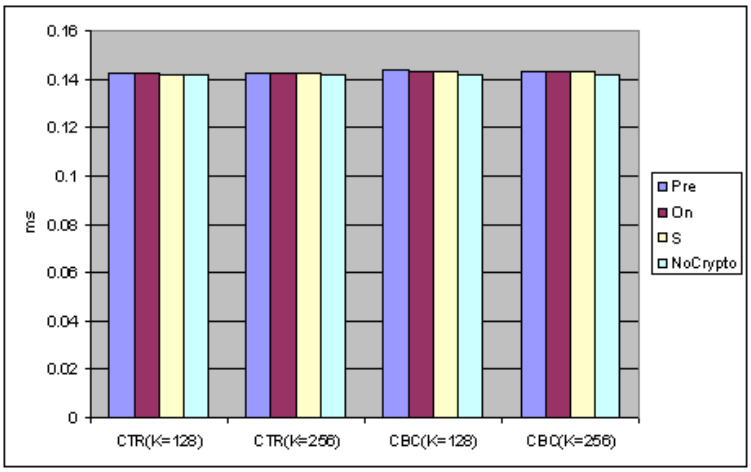
Figure 5.12 presents the simulation results for CBC and CTR schemes, where the legend represents the postfix of the schemes (if needed). For CBC schemes the encryption overhead ranges from 5% to 15% for small packets and for large packets the encryption overhead is considered negligible. For CTR schemes the encryption overhead ranges from 3% to 7% for small packets and for large packets the encryption overhead is considered negligible. It is worth mentioning that CBC-S and CTR-S are considered the fastest schemes.

Table 5.9 presents the cryptographic overhead percentages associated with GSCM(x), GCM(x)-Pre and GCM(x)-On schemes when $\alpha = 40$. For more details on the simulation results refer to Appendix C.

Figure 5.13, presents the simulation results for the fastest GCM schemes. GSCM(64k) is considered the fastest scheme of GSCM(x), GCM(4k)-Pre is the fastest GCM(x)-Pre and GCM(4k)-On is the fastest GCM(x)-On. GSCM(64k)



(a) Simulation parameters (α=40 and UNIT=100,000).



(b) Simulation parameters (α=1500 and UNIT=100,000).

Figure 5.12: Network simulation: average time (in ms) needed to process a packet, using CBC and CTR schemes.

Table 5.9: Overhead percentages of GSCM(x), GCM(x)-Pre and GCM(x)-On schemes ($\alpha = 40$).

x	GCM(x)-Pre 128-bit	GCM(x)-Pre 256-bit	GCM(x)-On 128-bit	GCM(x)-On 256-bit	GSCM(x) 128-bit	GSCM(x) 256-bit
0	20	22	14	14	16	20
256	12	15	13	15	12	14
4k	13	14	9	15	10	10
8k	11	15	16	16	6	9
64k	45	58	48	50	4	8

is faster than GCM(4k)-Pre and GCM(4k)-On when $\alpha = 40$, where it is about 7% faster than GCM(4k)-Pre, and about 8% faster than GCM(4k)-On. It is worth mentioning that for large packets all the schemes have almost the same speed.

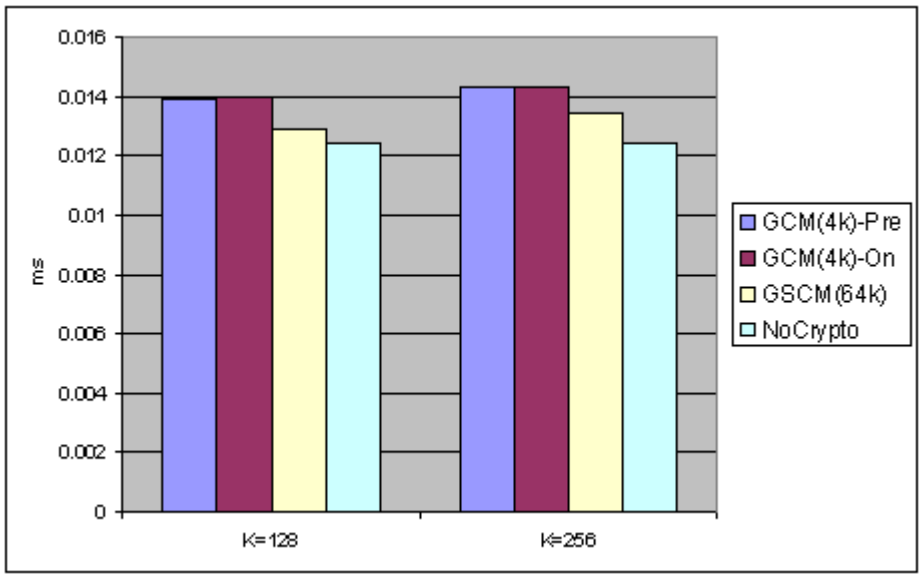


Figure 5.13: Network simulation: average time (in ms) needed to process a packet ($\alpha = 40$) by the fastest GCM schemes.

5.11 Security Analysis of the Schemes

5.11.1 Security of CBC and CTR

Birthday attacks on CBC and CTR modes remain possible even when the underlying block cipher is ideal [7], and CBC/CTR encryption becomes insecure once 2^{64} (in case of AES) blocks have been encrypted, in the sense that at this point partial information about the message begins to leak, due to birthday attacks [8]. Therefore, the server MUST generate a fresh key (random and unused key) before 2^{64} blocks are encrypted with the same key for each client. It is recommended to encrypt a maximum of 2^{32} blocks for each client (which is sufficient to encrypt the largest possible IPv6 jumbogram [26]), and then to generate a fresh key for that client to avoid birthday attacks.

5.11.2 Security of AES

It is assumed that, the best known attack against AES is to try every possible 128-/256-bits key (i.e. to perform an exhaustive key search), this requires about $2^{127}/2^{255}$ trails.

5.11.3 The Security of Even-Mansour Construction

Even-Mansour Construction is defined by (5.3), where **F** is a publicly known permutation, **P** is the plaintext, **K1** and **K2** are the keys of the constriction and **C** is the ciphertext. **P**, **C**, **K1** and **K2** are all n-bit strings.

$$C = F(P \oplus K1) \oplus K2 \tag{5.3}$$

The security of Even-Mansour construction [58] (when AES is used) is:

1. About 2^{255}, using exhaustive search over the key space, which is considered large enough by today's standards.

2. Daemen demonstrated in [34] that a known plaintext attack will take on average 2^{127} calculations, which has the same complexity as attacking AES with 128-bits key and is considered secure by today's technology.

3. Daemen also demonstrated in [34] that a chosen plaintext attack will take on average 2^{64} calculations using 2^{64} stored blocks. By limiting the number of encrypted blocks per client, this attack can be avoided (refer to Section 5.11.1 , as each client encrypts a maximum of 2^{32} blocks using the same secondary key. For some applications where more data is needed to be encrypted, the client can join a new cluster "using a fresh secondary key" or new fresh secondary key can be generated for that client).

4. Biryukov-Wagner demonstrated in [22] that a "sliding with a twist" attack allows an adversary to recover the key using $\sqrt{2} \times 2^{64}$ known plaintexts and $\sqrt{2} \times 2^{64}$ computations. By limiting the number of blocks encrypted per client using the same secondary key, this attack can be avoided (refer to Section 5.11.1).

5.11.4 Security of the Schemes

The security of CTR-Pre, CTR-On, GCM(x)-Pre and GCM(x)-On schemes are inherited from the security of CTR mode and that of AES, and the security of CBC-Pre and CBC-On is inherited from the security of CBC mode and that of AES, as the adversary can either attack the mode of operation or the cipher itself. The server should generate a fresh key (for each client) before 2^{64} blocks are encrypted with the same key, to avoid birthday attacks on CBC and CTR. It is recommended to encrypt a maximum of 2^{32} blocks for each client (which is sufficient to encrypt the largest possible IPv6 jumbogram [26]), then to generate a fresh key for that client.

The security of CTR-S/GSCM(x) schemes is inherited from that of CTR mode and that of AES2S and the security of CBC-S scheme is inherited from that of CBC mode and that of AESS2. There are two kinds of adversaries on the proposed schemes, when attacking AES2S/AESS2 (key search attack):

1. An outside adversary **A** (not a client served by the server) who watches the ciphertext. For an external adversary (who does not possess the primary key), she needs to attack AES with two random round keys.

2. An insider **B** (most probably a current user or a recently revoked user from the cluster) who would like to attack another client in the same cluster. This adversary knows the primary key:

(a) For CTR-S/GSCM(x) schemes:

- The adversary decrypts the known ciphertext with the known primary key until the 7^{th} or 10^{th} round (depending on the primary key size) to produce the intermediate state γ.
- By knowing the input to AES2S, the adversary can reduce AES2S to an Even-Mansour construction [58], where K1 and K2 are the keys and the reduced AES (7 or 10 rounds) is considered as the pseudorandom permutation.

(b) For CBC-S scheme:

- The adversary decrypts the known ciphertext with the known primary key until the 7^{th} or 10^{th} round (depending on the primary key size) to produce the intermediate state γ.
- The adversary encrypts the known plaintext with the known primary key until the $3^{rd}/4^{th}$ round (without the AddRoundKey operation) to produce the intermediate state ϕ.
- Now, the adversary has managed to reduce AESS2 to an Even-Mansour construction [58], where secondary key is the key of the construction and the reduced AES (4/6 rounds) is considered as the pseudorandom permutation.

Note that it was evident that AES has a random profile after only 3 rounds [151].

The most powerful adversary is **B**, where an insider adversary attack a client in the same cluster. The complexity to mount Daemen's known plaintext attack is the same complexity to attack AES with 128-bits key, which is considered secure with today's technology. On the other hand, to limit the probability of the other attacks, the number of encrypted blocks per client (using the same secondary key) MUST NOT reach the 2^{64} boundary. Therefore the server MUST generate a fresh key (for each client) before 2^{64} blocks are encrypted with the same key. It is recommended that the server encrypts 2^{32} blocks maximum for each client. For some applications where more data is needed to be encrypted, the client can join a new cluster or new fresh secondary key can be generated for that client.

The security CTR-S and GSCM(x) are upper bounded with the security of AES2S (refer to Sect. 4.9.6) and lower bounded with the security of Even-Mansour (refer to Sect. 5.11.3), and the security CBC-S is upper bounded with the security of AESS2 (refer to Sect. 4.9.5) and lower bounded with the security of Even-Mansour (refer to Sect. 5.11.3).

5.12 Summary

In this chapter, three novel encryption schemes for network applications are proposed. The main idea of the schemes is that each client possesses two keys. The first key is the cluster key, which is shared among the clients in the same cluster. The second key is the client's session key, which is unique for each client and is used to distinguish the clients. The experimental analysis demonstrates that the proposed schemes are superior to the state of the art schemes. Finally, the security analyses of the proposed schemes are presented.

Chapter 6

Disk Encryption

6.1 Introduction

Disk encryption applications usually encrypt/decrypt a whole sector at a time. There exist dedicated block ciphers that encrypt a sector at a time like Bear and Lion [5]. These ciphers are considered to be slow, as they process the data through multiple passes. Another method is to let a block cipher like the Advanced Encryption Standard (AES) [121] (with 16 bytes block size) to process the data using a mode of operation. The modes of operation can be divided into two main classes: the narrow-block and wide-block modes. The narrow-block modes operate on relatively small portions of data (typically 16 bytes when AES is used), while the wide-block modes encrypt or decrypt a whole sector (typically 512 bytes) at a time [134].

6.1.1 Disk Encryption Constraints

The main constraints facing disk encryption applications are:

Data size: the ciphertext length should be the same as the plaintext length. The current standard (512-bytes) will be used for the plaintext.

Performance: the used mode of operation should be fast enough, as to be transparent to the users (If using the mode of operation results in a significant and noticeable slowdown of the computer there will be great user resistance to its deployment [60]).

6.1.2 General Scheme

Figure 6.1 presents the general scheme for encrypting a sector, where a mode of operation takes four inputs to compute the ciphertext (4096-bit). These inputs are:

1. The plaintext of size 4096-bit.
2. Encryption key of size 128- or 256-bit.
3. Tweak Key of size 128- or 256-bit.
4. Sector ID of size 64-bit.

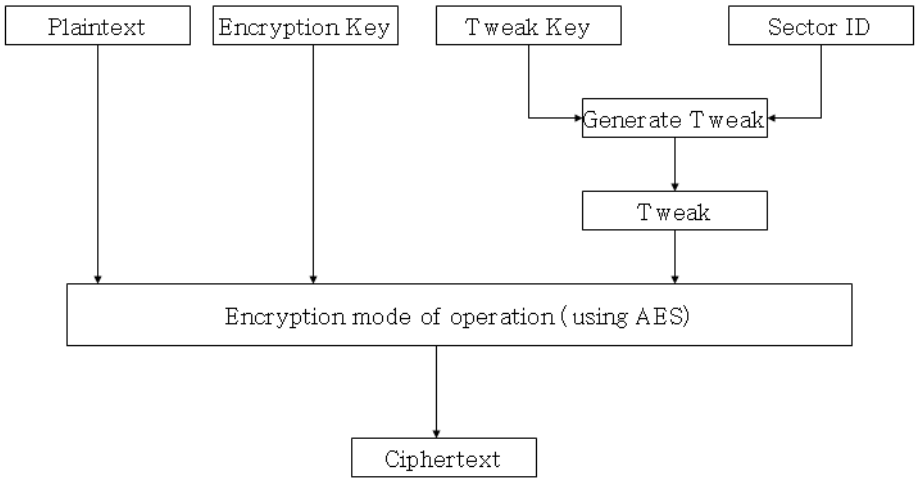


Figure 6.1: General scheme for encrypting a sector.

6.1.3 Tweak

Usually a block cipher accepts the plaintext and the encryption key to produce the ciphertext. Modes of operations have introduced other inputs. Some of these modes use initialization vectors like in CBC, CFB and OFB modes [117]; counters like in CTR [113] or nonces like in OCB mode [142]. The idea of using a tweak was suggested in HPC [146]. In [105], the formal definition of tweakable block ciphers has been introduced.

6.1.4 Tweak Calculation

There are different methods to calculate the tweak from the sector ID like ES-SIV [62] and the encrypted sector ID [60]. The encrypted sector ID approach will be used, where the sector ID (after being padded with zeros) is encrypted by the tweak key to produce the tweak.

6.1.5 Attack Models

In the analysis, two attack models are used: an active attack model and a less restrictive attack model. The adversary succeeds to break a mode of operation, if she can recover the used encryption keys or remove their effect.

Active Attack Model

It is assumed that the adversary has access to the encrypted hard disk, can read and modify the ciphertext stored on the hard disk, can ask the disk encryption application to encrypt some sectors for her, and force the hard disk encryption application to re-encrypt a sector for her. This can be done if the adversary can modify the plaintext of the sector in memory and ask the disk encryption application to save it.

A Less Restrictive Attack Model

This attack models is similar to the active attack model, with the exception that the adversary cannot force the hard disk encryption application to re-encrypt a sector for her.

6.2 Current Modes of Operations

6.2.1 Terminologies

The following terminologies are used to describe the modes of operation in this chapter:

IN: the input plaintext of size 4096-bit.

EKey: the encryption key of size 128 or 256-bit.

TKey: the tweak key of size 128 or 256-bit.

MKey: the mask key of size 128 or 256-bit.

SID: the sector ID encoded as an unsigned integer.

GetTweak(TKey,SID): encrypts (using AES) **SID** after padding with zeros with **TKey** and returns the result.

T: the tweak.

ExKey: the expanded AES key.

Expand-Key(EKey): expands the **EKey** with AES key setup routine and returns the result.

$\mathbf{X}_i$**:** the i^{th} block of text **X**, where a block is 128-bit.

$\bigoplus$**:** bitwise xor operation.

$\bigotimes$**:** multiplication operation in finite field.

OUT=Encrypt-AES(ExKey,IN): encrypts **IN**, using AES with **ExKey** as the expanded key, and returns **OUT**.

Substitute(T,ExKey,i): replaces the i^{th} round key in **ExKey** with **T** (Note that: the first round of AES is round zero and it is the pre-whitening process).

len(X): returns the length of the string **X** in bits.

<<: is a left rotation operation, where the rotation value is written on its right size.

6.2.2 Cipher Block Chaining (CBC)

CBC was the most used encryption mode of operation for hard disk encryption applications [62], where usually each sector is encrypted with a different initialization vector (IV). The listing of CBC is in table 6.1. For the first block the plaintext is xored with the tweak (which serves as the initialization vector **IV**) before it is encrypted, and for the other blocks: the plaintext is xored with the last ciphertext block before it is encrypted. This recursive structure does not allow parallelization (in encryption direction).

Table 6.1: CBC listing for disk encryption.

```
Encrypt-CBC(IN,EKey,TKey,SID)
    T=GetTweak(TKey,SID)
    ExKey=Expand-Key(EKey)
    IN_0= IN_0 ⊕ T
    OUT_0=Encrypt-AES(ExKey,IN_0)
    for i=1 to 31
        IN_i= IN_i ⊕ OUT_{i-1}
        OUT_i=Encrypt-AES(ExKey,IN_i)
    end for
return OUT
```

For CBC, an adversary with read/write access to the encrypted disk can copy a ciphertext sector from one position to another, and an application reading the sector of the new location will still get the same plaintext sector (except perhaps the first 128-bit) [134]. In CBC, the adversary can flip arbitrary bits in one block at the cost of randomizing the previous block [60].

Due to these weaknesses, it is not recommended using CBC in disk encryption applications.

6.2.3 LRW

LRW is a tweakable mode of operation that can be parallelized. The listing of LRW is in table 6.2. It xors the text before and after encryption with the sector key (Note that the sector key value is multiplied "in the finite field of AES" with a counter).

Table 6.2: LRW listing for disk encryption.

```
Encrypt-LRW(IN,EKey,TKey,SID)
    ExKey=Expand-Key(EKey)
    T=TKey
    for i=0 to 31
        T = T ⊗ SID
        IN_i = IN_i ⊕ T
        OUT_i=Encrypt-AES(ExKey,IN_i)
        OUT_i= OUT_i ⊕ T
        SID=SID +1
    end for
return OUT
```

The IEEE Security in Storage Work Group (SISWG) [1] used LRW in its

early draft as a standard for narrow-block encryption mode (where AES is the underlying cipher). Then it was replaced by XTS [141] due to the following security issue. An adversary can derive LRW sector key **TKey** from the ciphertext if the plaintext contains **TKey**$||0^n$ or $0^n||$**TKey**. Here $||$ is the concatenation operation and 0^n is a block with n zeros. This may be an issue for software that encrypts the partition of an operating system under which this encryption software is running (at the same time). The operating system could write **TKey** to encrypted swap/hibernation file. If **TKey** is known, LRW does not offer indistinguishability under chosen plaintext attack anymore, and the block permutation attacks of ECB mode are possible [73]. Due to this weakness, it is not recommended using LRW in disk encryption applications.

6.2.4 XTS

XTS is a tweakable mode of operation that can be parallelized. The listing of XTS is in table 6.3. It xors the plaintext before and after encryption with the tweak value (Note that the tweak value is multiplied in the finite field GF(2^{128}) in each loop with the ith power of α "a primitive element of GF(2^{128})").

Table 6.3: XTS listing for disk encryption.

```
Encrypt-XTS(IN,EKey,TKey,SID)
    T=GetTweak(TKey,SID)
    ExKey=Expand-Key(EKey)
    for i=0 to 31
        X = T ⊗ α^i
        IN_i = IN_i ⊕ X
        OUT_i=Encrypt-AES(ExKey,IN_i)
        OUT_i= OUT_i ⊕ X
    end for
return OUT
```

XTS is now the current standard of SISWG for the narrow-block modes of operation. XTS is an instantiation of XEX [141] mode of operation and this mode of operation has a problem when a lot of data is encrypted with the same key, because a collision will appear [134].

There is a serious concern, due to the restrictions on the number of encryption operations. The P1619 standard provides good security bounds as long as: "the same key is not used to encrypt much more than a terabyte of data". This is not the capacity of the storage device, but the number of encryption operations. In many applications this limit is reached in just hours, requiring partitioning the data into thousands of small bands (using different keys). This solution is very expensive and slows down the system [98].

It was also shown in [130] that the terabyte limit is easily reached using "*Temporal effects*". For instance, on a disk with 4 KB blocks where each block is a data unit, an adversary who observes approximately 4000 writes to a given block will have obtained access to 2^{20} cipher blocks with the same tweak and key. Similarly, an adversary who steals a 500 GB encrypted disk may get access to about 1 TB of ciphertext if she was also able to recover the previous contents of each sector.

When the number of blocks encrypted by XTS approach 2^{64} (the birthday boundary), there is a non-negligible probability that for some *i,j* there is a collision of the form shown in (6.1).

$$P_i \oplus T_i = P_j \oplus T_j \tag{6.1}$$

This implies that:

$$C_i \oplus T_i = E(P_i \oplus T_i, K) = E(P_j \oplus T_j, K) = C_j \oplus T_j \tag{6.2}$$

Summing the above two equations implies:

$$P_i \oplus C_i = P_j \oplus C_j \tag{6.3}$$

This can be used to distinguish XTS from a collection of truly random permutations. The adversary computes for all i the sum $S_i = P_i \oplus C_i$ and counts the number of pairs (i,j) for which $S_i = S_j$. The argument above implies that for any *i,j* , the probability that $S_i = S_j$ in ciphertext produced by XTS is roughly $2^{-n} + 2^{-n} = 2^{-n+1}$, where the first term is due to collision (between i and j) and the second term is due to equality $S_i = S_j$ without collision.

In case of a collision between block i and block j, the adversary can use her ability to create legally encrypted data for position i and her ability to modify ciphertext in position j, to modify the ciphertext at block j so it will decrypt to an arbitrary adversary-controlled value. The adversary encrypts a new value $P'_i = P_i \oplus \Delta$, for some $\Delta \neq 0$, and observes the ciphertext C'_i, the adversary now replaces the ciphertext block C_j by:

$$C'_j = C_j \oplus (C_i \oplus C'_i) \tag{6.4}$$

This new ciphertext will be decrypted as $P'_j = P_j \oplus \Delta$, in other words the adversary succeeded in *flipping* specific bits in the plaintext corresponding to location j.

There is also a weak key issue if the tweak value ever starts at zero. In this case, there is no masking and the security reduces to that of ECB mode. However, the probability of this occurrence is 2^{-128}, which is reasonable for practical encryption applications [108].

Due to these weaknesses, it is not recommended using XTS in disk encryption applications.

6.3 Masked Code Book (MCB)

6.3.1 Design Goals

The design goals of MCB mode are:

Security: The constraints for disk encryption imply that the best achievable security is essentially what can be obtained by using ECB mode with a different key per block [134], which is the aim.

Performance: MCB should have a comparable performance to the state of the art modes of operation.

Parallelization: In today's world of multiple core processors and clock-rate limitations, it is increasingly important that a designer is able to increase performance by instantiating multiple instances of an encryption primitive instead of increasing the clock rate of an existing encryption primitive. MCB should operate in parallel, allowing scalability in today's environment.

6.3.2 Keys of MCB

The encryption key of MCB is divided into three parts (each of them of size 128- or 256-bit):

1. **EKey**: this key is used for encryption.
2. **TKey**: this key is used to encrypt the sector ID to produce the tweak.
3. **MKey**: this key is used to generate the mask (**M**).

6.3.3 The Mask

The mask (**M**) is constructed once at the initialization of MCB. It is an array of sixty four 128-bit blocks. It is constructed using AES in the counter mode (CTR) [113], where the counter is initialized with zero and **MKey** is the encryption key for the counter mode.

6.3.4 Design

The constraints for disk encryption imply that the best achievable security is essentially what can be obtained by using ECB mode with a different key per block [134]. This is achieved by MCB mode as each block is encrypted using different masks. The listing of MCB mode is in table 6.4 and it works as follows:

1. The tweak **T** is calculated by encrypting the sector ID with the tweak key.
2. The expanded key **ExKey** is calculated.
3. A unique mask (**M1**) is constructed by xoring the tweak with the first 32 blocks of the mask **M**.
4. The plaintext is xored with **M1**.
5. The result of the previous step is encrypted using the ECB mode.
6. A unique mask (**M2**) is constructed by xoring the tweak with the last 32 blocks of the mask **M**.
7. The ciphertext (result from the ECB mode) is xored with **M2**, and the result is returned as ciphertext.

In table 6.5, MCB is rewritten using the proposed AES variant AES2I (refer to Sect. 4.7.1).

Table 6.4: MCB listing for disk encryption.

```
Encrypt-MCB(IN,EKey,TKey,SID)
    T=GetTweak(TKey,SID)
    ExKey=Expand-Key(EKey)
    for i=0 to 31
        M1_i = T ⊕ M[i]
        IN_i = IN_i ⊕ M1_i
        OUT_i = Encrypt-AES(ExKey,IN_i)
        M2_i = T ⊕ M[32+i]
        OUT_i = OUT_i ⊕ M2_i
    end for
return OUT
```

Table 6.5: MCB listing for disk encryption (using AES2I).

```
Encrypt-MCB(IN,EKey,TKey,SID)
    T=GetTweak(TKey,SID)
    ExKey=Expand-Key(EKey)
    for i=0 to 31
        M1_i = T ⊕ M[i]
        M2_i = T ⊕ M[32+i]
        OUT_i = Encrypt-AES2I(ExKey,IN_i,M1_i,M2_i)
    end for
return OUT
```

6.3.5 Security of MCB

The goal of MCB is to encrypt each block on the hard drive in a different way, which is the best achievable security for disk encryption applications (with current constraints) [134]. This was achieved by the followings:

1. Using the pre-whitening masking step, assures that encrypting the same plaintext blocks (within the same sector) will result in completely different ciphertext blocks (as each block in M1 is unique).

2. Using the post-whitening masking step, assures that decrypting the same ciphertext blocks (within the same sector) will result in completely different plaintext blocks (as each block in M2 is unique).

MCB mode operates like ECB mode, but:

- By introducing the tweak, the adversary cannot perform the mix-and-match attack [134] among blocks of different sectors, as each sector has a unique tweak. Thus, encrypting the same block in different sectors will produce two different ciphertexts and decrypting the same ciphertext in different sectors will produce different plaintexts.

- By introducing the mask **M** the adversary cannot perform the "mix and match" attack among the blocks within the same sector. As each block has its unique mask.

- The above two properties eliminate the main weakness of ECB.

MCB is secure as long as its three keys are not known to the adversary. Below it is discussed, how the adversary can benefit from knowing one or two of these three keys:

Case EKey is known: MCB is reduced to an Even-Mansour construction [58], where **M1** and **M2** are the keys and AES is considered as the pseudorandom permutation. But as **M1** and **M2** are unique for each sector, a practical attack on Even-Mansour construction is not feasible.

Case MKey is known: The adversary can compute the mask **M**. The adversary can xor the first 32 blocks of **M** to the plaintext and the last 32 blocks of **M** to the ciphertext to remove their original effects. Now the adversary can perform the "mix and match" attacks to the blocks within the same sector.

Case TKey is known: The adversary can compute the tweak **T**. The adversary can xor **T** to the plaintext and the ciphertext to remove its original effects. Now the adversary can perform the "mix and match" attacks to whole sectors.

Case EKey and MKey are known: MCB is reduced to an Even-Mansour construction, where **T** is the key and AES is considered as the pseudorandom permutation. But as **T** is not known (which is unique for each sector), a practical attack on Even-Mansour construction is not feasible.

Case EKey and TKey are known: MCB is reduced to an Even-Mansour construction, where **M** is the key and AES is considered as the pseudorandom permutation. After encrypting 2^{64} sectors (2^{73} bits), the attacks on Even-Mansour construction are applicable.

Case MKey and TKey are known: MCB is reduced to ECB and the "mix and match" attacks are applicable.

For the security of Even-Mansour construction refer to Sect. 5.11.3. As the mask is unique for each block, those attacks are considered not practical.

6.3.6 Security Against Ciphertext Collisions

A collision happens, when $C_i^x = C_j^y$ where C_k^w denotes the ciphertext block with index k in the sector number w.

Recall that a block encrypted by MCB is defined as:

$$C_i^x = E(P_i^x \oplus M1_i^x, K) \oplus M2_i^x \tag{6.5}$$

Proposition 6.1. *The pre-whitening mask of a block in MCB is not equal to the post-whitening mask of that block, in other words:* $M1_i^x \neq M2_i^x$.

Proof.

$$M1_i^x = M1_i \oplus T^x \tag{6.6}$$
$$M2_i^x = M2_i \oplus T^x \tag{6.7}$$
$$\tag{6.8}$$

Summing the above two equations:

$$M1_i^x \oplus M2_i^x = M1_i \oplus M2_i \tag{6.9}$$

Since the right hand side can never be zero ($M1_i$ and $M2_i$ are generated using the counter mode, with the same key, and since AES is a deterministic cipher, thus $M1_i$ can never have the same value as $M2_i$), therefore the left hand side can never be zero (i.e. $M1_i^x \neq M2_i^x$). □

Corollary 6.3.1. *In case of a collision in ciphertext, this implies that the plaintexts are not equal.*

Proof. If the collision is in the same sector, this means that the used masks are different thus the plaintexts are different.
If the collision is in different sectors, this implies that the used tweaks are different thus the plaintexts are different. □

Corollary 6.3.2. *There is no tweak that can reduce the security of a sector to that of ECB mode.*

Proof. From the above proposition $M1^x \neq M2^x$, so if $M1^x = 0$ then $M2^x \neq 0$ and vice versa. □

6.3.7 Advantages of MCB

Security: As each sector is encrypted in a different way, the best achievable security for disk encryption applications (with current constraints) [134] is met.

Performance: MCB possesses high performance as it uses only simple and fast operations.

Parallelization: MCB can be easily parallelized, as each block is independent on the other blocks.

MCB meets all its design goals.

6.4 Substitution Cipher Chaining Mode (SCC)

6.4.1 Design Goals

The design goals of SCC mode are:

Security: SCC is designed to modify Windows Vista's Disk Encryption algorithm (ELEPHANT), refer to Sect. 6.5, and it is not designed to be used as a stand alone disk encryption mode of operation.

Performance: SCC should have a comparable performance to the state of the art modes of operation.

Parallelization: SCC should offer some kind of parallelization.

Error propagation: SCC should offer error propagation.

6.4.2 Keys

The secret key in SCC is divided into three different keys (each of them can be either 128- or 256-bit):

1. **EKey**: This key generates the expanded key, used in encrypting the blocks.
2. **TKey**: This key encrypts the sector ID to produce the tweak.
3. **MKey**: This key generates the mask **M**, where **M** is an array of sixty four 128-bit blocks. **M** is constructed once at the initialization of SCC mode, it is constructed using AES in the counter mode (CTR) [113], where the counter is initialized with zero and **MKey** is the encryption key for the counter mode.

6.4.3 Design

SCC mode is built using SSM model [47] to inherit from its high performance and uses CBC like operations to gain the error propagation property. The listing of SCC is in table 6.6 and it works as follows:

Table 6.6: SCC listing for disk encryption.

```
Encrypt-SCC(IN,EKey,Keylen,TKey,SID)
    T=GetTweak(TKey,SID)
    ExKey=Expand-Key(EKey)
    KL=len(EKey)
    if(KL==128)
        x=4    y=5    z=6
    else
        x=5    y=7    z=10
    end if
    Substitute(T,ExKey,y)
    Substitute(α0,ExKey,x)
    Substitute(β0,ExKey,z)
    OUT0=Encrypt-AES(IN0,ExKey)
    for i=1 to 31
        Substitute(αi,ExKey,x)
        Substitute(βi,ExKey,z)
        τ=OUTi−1 ⊕ T
        Substitute(τ,ExKey,y)
    OUTi=Encrypt-AES(INi,ExKey)
    end for
return OUT
```

where $\alpha_i = M_{2\times i}$ and $\beta_i = M_{(2\times i)+1}$

- The tweak **T** is calculated by encrypting the sector ID with the tweak key **TKey**.
- The expanded key **ExKey** is calculated.
- The values of x, y and z are determined by the encryption key size.

- For the first block, the secret tweak **T** replaces the round key of the y^{th} round, the secret 128-bit α_0 replaces the round key of the x^{th} round, and the secret 128-bit β_0 replaces the round key of the z^{th} round. Finally, the first block is encrypted by the new expanded key.
- A loop that runs 31 times (where i takes the values from 1 to 31), the secret 128-bit α_i replaces the round key of the x^{th} round, the secret 128-bit β_i replaces the round key of the z^{th} round, and τ replaces the round key of the y^{th} round (where τ is calculated by xoring ciphertext of the previous block with T and it acts as the *active tweak*). Finally, the i^{th} block is encrypted by the new expanded key.

6.4.4 Advantages of SCC

Performance: SCC possesses high performance as it uses only simple and fast operations.

Parallelization: SCC can be parallelized, actually it favors dual core processors. It can be parallelized as following (using two processor cores):

- The second processor will compute till the y^{th} round for all the plaintext blocks (except the first block), to produce intermediate ciphertext blocks.
- The first processor will compute the first ciphertext block from the first plaintext block.
- The first processor will compute the other ciphertext blocks from intermediate ciphertext blocks processed by the second processor.

In this way the ciphertext will be produced after 16.5 encryptions.

Error propagation: As each sector depends on its previous sector, error propagation is met.

SCC meets all its design goals.

6.4.5 Security of SCC

As mentioned in Sect. 6.4.1, SCC is designed to modify Windows Vista's Disk Encryption algorithm "ELEPHANT" (refer to Sect. 6.5), and it is not designed to be used as a stand alone disk encryption mode of operation.

The reason that SCC should not be used as a stand alone disk encryption mode of operation is that the adversary can manipulate the *active tweak* τ, thus it does not meet the guidelines in Sect. 4.9 and is not considered secure. In fact, to demonstrate the lack of security of SCC, it is broken in Sect. 6.8 using new proposed attacks. For this reason, it is not recommended to use SCC for disk encryption applications.

6.5 ELEPHANT

Windows Vista Enterprise and Ultimate editions use Bitlocker Drive Encryption as its disk encryption algorithm, and at its heart is AES-CBC + Elephant

diffuser encryption algorithm (ELEPHANT). Bitlocker uses existing technologies like AES in CBC mode and TPM [155], together with two new diffusers. Figure 6.2 presents an overview of ELEPHANT [60]. There are four steps to encrypt a sector:

1. The plaintext is xored with a sector key K_s (6.10).
2. The result of the previous step runs through diffuser A.
3. The result of the previous step runs through diffuser B.
4. The result of the previous step is encrypted with AES in CBC mode using IV_s (6.11), as the initialization vector.

$$K_s = E(K_{sec}, e(s)) \parallel E(K_{sec}, e'(s)) \tag{6.10}$$

$$IV_s = E(K_{AES}, e(s)) \tag{6.11}$$

Where E() is AES encryption function, K_{sec} is a key used to generate K_s, K_{AES} is the key used to generate the sector IV_s and used in AES-CBC process, e() is an encoding function that maps each sector number s into a unique 16-byte value. The first 8 bytes of the result are the byte offset of the sector on the volume. This integer is encoded in least-significant-byte first encoding. The last 8 bytes of the result are always zero and e′(s) is the same as e(s) except that the last byte of the result has the value 128.
Note that the plaintext and key are parameterized. In this thesis, the following parameters are used:

1. Plaintext of size 4096-bit (the current standard sector size).
2. Both 128-bit and 256-bit key variants of AES are examined, which means both K_{sec} and K_{AES} are either 128-bit or 256-bit.

6.5.1 The Diffusers

Diffuser A and diffuser B are very similar. The following notations are used to define the diffusers:

1. $\mathbf{d}_i$ is the i^{th} 32-bit word in the sector, if i falls outside the range then $\mathbf{d}_i = \mathbf{d}_{i \bmod n}$, where **n** is the number of the 32-bit in the sector.
2. **AC** and **BC** are the number of cycles of diffuser A and B, they are defined to be 5 and 3 respectively.
3. **RA** = [9, 0, 13, 0] and **RB** = [0, 10, 0, 25] hold the rotation constants of diffuser A and B respectively.
4. $\oplus$ is the bitwise xor operation.
5. $<<$ is the integer 32-bit left rotation operation, where the rotation value is written on its right size.
6. - is integer subtraction modulo 2^{32}.

Table 6.7 presents the description of diffuser A and diffuser B.

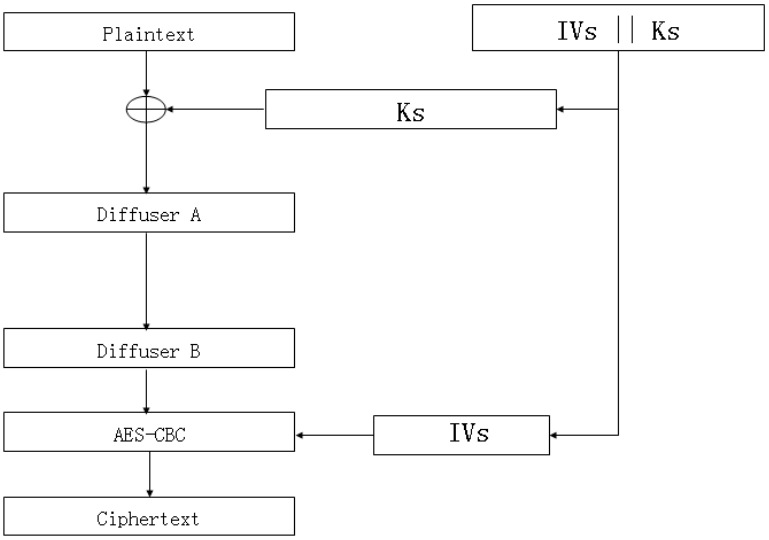


Figure 6.2: Overview of AES-CBC with Elephant Diffuser.

Table 6.7: Diffuser A and diffuser B.

Diffuser A:	**Diffuser B:**
for j=1 to AC	for j=1 to BC
for i=n-1,...,2,1,0	for i= n-1,...,2,1,0
t=(d_{i-5} << $RA_{i \bmod 4}$)	t=(d_{i+5} << $RB_{i \bmod 4}$)
t=t$\oplus d_{i-2}$	t=t$\oplus d_{i+2}$
d_i=d_i-t	d_i=d_i-t

6.5.2 Proposed Modification

It is proposed to replace CBC layer in ELEPHANT with an SCC layer. The names of these variants are ELEPHANT$^+$ (when **AC**=5 and **BC**=3) and ELEPHANT* (when **AC**=**BC**=3). The advantages of this modification over the original design are:

- The SCC can be parallelized on a dual core machine, which can increase the performance.
- SCC has better error propagation than CBC in decryption direction, as more bits of plaintext will be affected by a single bit change.
- These proposed variants are faster than ELEPHANT.
- These proposed variants possess a higher Safety Factor than ELEPHANT (refer to Sect. 6.5.4).

6.5.3 Bit Dependency Tests

The bit dependency tests are developed to measure the minimum values required for **AC** and **BC** to achieve full diffusion.

- **BD-Encryption()**: is passed, when each bit in the ciphertext depends on every bit in the plaintext.
- **BD-Decryption()**: is passed, when each bit in the plaintext depends on every bit in the ciphertext.

The Bit-dependency functions are measured as following:

1. A dependency matrix **D** is constructed of size B $\times$ B (where B is the number of bits in the plaintext/ciphertext, here B = 4096).
2. The diagonal is initialized by 1 and all other bits are set to zero, as initially each bit depends only on itself.
3. Depending on the applied operation the matrix **D** is updated, BD-Encryption applies the operation in the encryption direction and BD-Decryption applies them in the decryption direction.
4. If an output bit is dependent on an input bit(s), the column of the output bit is ORed with that (those) of the input bit(s). For example:
   (a) Xor operation: each output bit is dependent on the corresponding input bit.
   (b) Addition and subtraction modulo 2^{32} operations are approximated to xor operation for simplicity and generality.
   (c) AES operation using CBC: each bit in the input 128-bit is dependent on the other 127-bit.
   (d) AES operation using SCC: each bit in the previous block (with the exception of the first block) affects all the bits of the encrypted block (as full confusion and full diffusion properties are met).
   (e) 32-bit rotation: the columns change their order depending on the rotation amount and direction.
5. All the operations of the tested ciphers are applied and the matrix **D** is updated.
6. At the end the sum of all ones in the matrix is calculated and is divided by B $\times$ B.
7. If the returned value in the previous step is 1, this means that each bit of the output bits depends on all the bits of the input and the function is passed, it fails otherwise.

The results of applying **BD-Encryption** and **BD-Decryption** functions are found in table 6.8, where the minimum values of AC and BC required by each algorithm to pass these tests are reported (under columns AC′ and BC′), together with the used values. These results show that ELEPHANT needs at least AC=2 and BC=1 to pass BD-Encryption and BD-Decryption functions, while ELEPHANT* and ELEPHANT$^+$ need only BC=2 to pass them and SCC layer does the rest of the diffusion.

Table 6.8: BD-Encryption and BD-Decryption results.

	AC′	**BC′**	**AC**	**BC**	**SF**
ELEPHANT	2	1	5	3	2.7
ELEPHANT$^+$	0	2	5	3	4
ELEPHANT*	0	2	3	3	3

6.5.4 Safety Factor

The Safety Factor (SF) is defined in (6.12), which is the ratio between the total number of used diffusers' cycles over the minimum required. SF represents how safe is the current number of diffusers' cycles, under any circumstances this ratio should not be less than one. The values of SF are reported in table 6.8. These values show that ELEPHANT$^+$ and ELEPHANT* possess a higher SF than ELEPHANT.

$$SF = (AC + BC) \div (AC' + BC') \tag{6.12}$$

6.6 Extended Substitution Cipher Chaining Mode (ESCC)

6.6.1 Design Goals

The design goals of ESCC mode are:

Security: The constraints for disk encryption imply that the best achievable security is essentially what can be obtained by using ECB mode with a different key per block [134], which is the aim.

Performance: ESCC should have a comparable performance to the state of the art modes of operation.

Error propagation: ESCC should propagate error to further blocks (this may be useful in some applications).

6.6.2 Keys

The secret key in ESCC is divided into three different keys (each of them can be either 128- or 256-bit):

1. **EKey**: This key generates the expanded key, used in encrypting the blocks.
2. **TKey**: This key encrypts the sector ID to produce the tweak.
3. **MKey**: This key generates the mask **M**, where **M** is an array of sixty four 128-bit blocks. **M** is constructed once at the initialization of ESCC mode, it is constructed using AES in the counter mode (CTR) [113], where the counter is initialized with zero and **MKey** is the encryption key for the counter mode.

6.6.3 Design

ESCC mode is built using SSM model [47] to inherit from its security and high performance, and uses CBC like operations to gain the error propagation property. The listing of ESCC is in table 6.9 and it works as follows:

Table 6.9: ESCC listing for disk encryption.

```
Encrypt-ESCC(IN,EKey,Keylen,TKey,SID)
    T=GetTweak(TKey,SID)
    ExKey=Expand-Key(EKey)
    KL=len(EKey)
    if(KL==128)
        x=4    y=5    z=6
    else
        x=5    y=7    z=10
    end if
    Substitute(T,ExKey,y)
    Substitute(M_0 ⊕ T,ExKey,x)
    Substitute(M_1 ⊕ T,ExKey,z)
    Encrypt-AES(ExKey,IN_0,OUT_0)
    for i=1 to 31
        Substitute(M_{2×i} ⊕ (OUT_{i-1} << 32),ExKey,x)
        Substitute(M_{2×(i+1)} ⊕ (OUT_{i-1} << 64),ExKey,z)
        TT=OUT_{i-1} ⊕ T
        Substitute(TT,ExKey,y)
        Encrypt-AES(ExKey,IN_i,OUT_i)
    end for
return OUT
```

- The tweak **T** is calculated by encrypting the sector ID with the tweak key **TKey**, due to this step the value of the tweak is neither known nor controlled by the adversary.
- The expanded key **ExKey** is calculated.
- The values of x, y and z are determined by the encryption key size.
- For the first block:
  - The secret tweak **T** replaces the round key of the y^{th} round.
  - The secret 128-bit $\mathbf{M}_0 \oplus T$ replaces the round key of the x^{th} round.
  - The secret 128-bit $\mathbf{M}_1 \oplus T$ replaces the round key of the z^{th} round.
  - The first block is encrypted by the new expanded key.
- A loop that runs 31 times (where i takes the values from 1 to 31):
  - The secret 128-bit $\mathbf{M}_{2\times i}$ xored with the ciphertext of the previous block after being rotated 32-bit to the left replaces the round key of the x^{th} round.

- The secret 128-bit $\mathbf{M}_{(2\times i)+1}$ xored with the ciphertext of the previous block after being rotated 64-bit to the left replaces the round key of the z^{th} round.
- A variable **TT** is calculated by xoring ciphertext of the previous block with T.
- **TT** acts as the *active tweak* and replaces the round key of the y^{th} round in the expanded key.
- The i^{th} block is encrypted by the new expanded key.

6.6.4 Discussion of ESCC

The goal of ESCC is to encrypt each block on the hard drive in a different way. This was achieved by using SSM model, where:

- The active tweak **TT** is placed in the middle of the expanded key to offer full diffusion and full confusion properties in both encryption and decryption directions (i.e. any difference between two active tweaks, will be associated with full confusion and full diffusion in both encryption and decryption directions, eliminating the bit-flipping attack of CBC mode). Note that AES requires only four rounds to obtain full bit confusion (or mixing) and diffusion (each input bit affecting each output bit) properties [112].
- Note that the active tweak **TT** is the result of xoring:
  1. The tweak **T** (which is unique, secret and not controlled by the adversary).
  2. The ciphertext of the previous block (which is known and controlled by the adversary).
  3. From the above two notes, the adversary does not know the value of **TT**, but can flip its bits. But by changing any bits of a ciphertext block, that yields to a difference in 3 different columns, which will destroy any attempt to mount chosen plaintext/ciphertext attacks as in Sect. 6.8, so any change in **TT** will be associated with full confusion and full diffusion in both encryption and decryption directions.
- Replacing the round key of the x^{th} and z^{th} rounds offers full diffusion and full confusion in encryption and decryption directions among the blocks of the same sector. Note that all the values of **M** are unique and key dependent.

Notes:

1. By introducing the tweak, the adversary cannot perform the mix-and-match attack [134] among blocks of different sectors, as each sector has a unique secret tweak. The tweak replaces the round key of the middle round of AES to assure that any difference between two tweaks will be associated with full confusion and full diffusion in both encryption and decryption directions. Thus, encrypting the same block in different sectors will produce two different ciphertexts and decrypting the same ciphertext in different sectors will produce different plaintexts.

2. By introducing the mask **M** (that replaces certain words in the expanded key) the adversary cannot perform the mix-and-match attack among the blocks within the same sector. As each sector has two distinct 128-bit in the expanded key. This requirement is achieved in both encryption and decryption directions. As equal plaintext blocks (within the same sector), will have the same state until the x^{th} encryption round then the state will change. Furthermore, equal ciphertext blocks (within the same sector), will have the same state until the z^{th} decryption round then the state will change.

6.6.5 Advantages of ESCC

Security: Each sector is encrypted in a different way, so replacing ciphertext between different sectors will not help the adversary, as they are encrypted with different expanded keys and each block within the sector is encrypted in a different way, due to the use of **M** (so the adversary will not benefit from changing the positions of the blocks).

Performance: ESCC possesses high performance as it uses only simple and fast operations.

Error propagation: As each sector depends on its previous sector, error propagation is met.

ESCC meets all its design goals.

6.7 Performance Analysis

To compare the performance of the proposed and standard modes of operation, a benchmark application is developed in C++. This application uses the following parameters:

Key: is either 128- or 256-bit. It represents the size of the encryption key used by AES.

Step: is either 512 bytes or 1 Megabyte. It represents the incremental difference between the measurement points.

Loops: is either 10 or 1000. It represents the number of measurements calculated for each point.

Test: the benchmark application measures the performance of four different tests:

Encrypt: measures the throughput if the mode of operation performs encryption in memory.

Decrypt: measures the throughput if the mode of operation performs decryption in memory.

Encrypt + Write: measures the throughput if the mode of operation performs encryption in memory and writes the result on the hard drive.

Read + Decrypt: measures the throughput if the mode of operation reads the data from the hard drive, then performs decryption in memory.

6.7.1 Benchmark Application

A benchmark application is developed; this application measures the performance at different measuring points. Each measuring point represents the size of data that will be processed. The values of these measuring points range from **STEP** to 200 × **STEP**. For each measuring point the data was processed **Loops** times and the mean was calculated. Then the mean of all measuring points was calculated to get the average processing speed for a particular mode of operation. In the benchmark, the performance of the Electronic Codebook (ECB) mode [117] is measured to be a reference for maximum throughput.

The configuration of the used computer in the benchmark is in table 6.10. The performance of the optimized C++ versions of the modes of operation are studied. For AES the optimized Gladman's implementation [66] is used, and for the diffusers (of ELEPHANT and its variants) loop unrolling mechanism [39] is used.

Table 6.10: Simulation Machine Configuration.

Processor	Intel Xeon Quad-Core 2.33 GHz (64-bit)
RAM	4096 MB
Processor Cache	12 MB
Paging file	4096 MB
OS	Microsoft Windows Server 2008
Compiler	Visual C++ 2008
Code optimization	Maximum speed

6.7.2 CPU Utilization

The CPU utilization is monitored, for the four **Tests** and the two different **Steps**. For encrypting or decrypting data in memory the CPU is fully utilized when processing large data (Step=1 MB) and about 80% utilized when processing small data (Step=512 byte). It is remarkable that, when the data is encrypted in memory and then written to the hard drive that the CPU utilization decreases to about 30% for large data and 70% for small data. Reading and decrypting data on the other hand utilizes the CPU by about 70% for small data and about 100% for large data.

6.7.3 Benchmarking Results

The results of the benchmark for large data (Step=1 MB and Key=256) are shown in Figure 6.3. From these results, it is shown that SCC is almost as fast as CBC and ESCC is about 10% slower than CBC. MCB possesses high throughput, as it is about 45% faster than LRW and 25% faster than XTS.

ELEPHANT^{+} is as fast as ELEPAHNT, while ELEPHANT* is faster than ELEPHANT by about 5%.

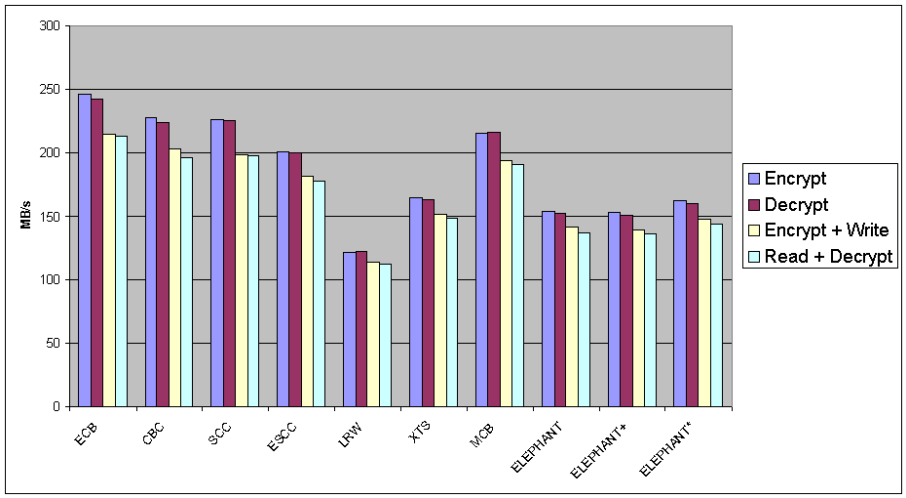


Figure 6.3: The average throughput of different modes of operation (Key=128, Step=512 byte, Loops=1000).

The results of the benchmark for small data (Step=512 and Key=256) are shown in Figure 6.4. From these results, it is shown that SCC is almost as fast as CBC and ESCC is about 8% slower than CBC. MCB possesses high throughput, as it is about 38% faster than LRW and 20% faster than XTS. ELEPHANT^{+} is as fast as ELEPAHNT, while ELEPHANT* is faster than ELEPHANT by about 5%.

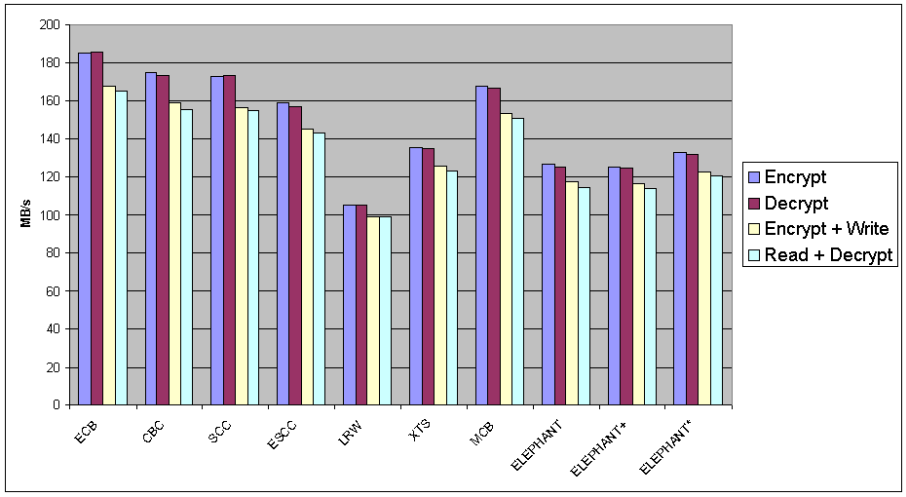


Figure 6.4: The average throughput of different modes of operation (Key=256, Step=512 byte, Loops=1000).

The results of the benchmark for large data (Step=1 MB and Key=128)

are shown in Figure 6.5. From these results, it is shown that SCC is almost as fast as CBC and ESCC is about 12% slower than CBC. MCB possesses high throughput, as it is about 48% faster than LRW and 27% faster than XTS. ELEPHANT$^+$ is as fast as ELEPAHNT, while ELEPHANT* is faster than ELEPHANT by about 5%.

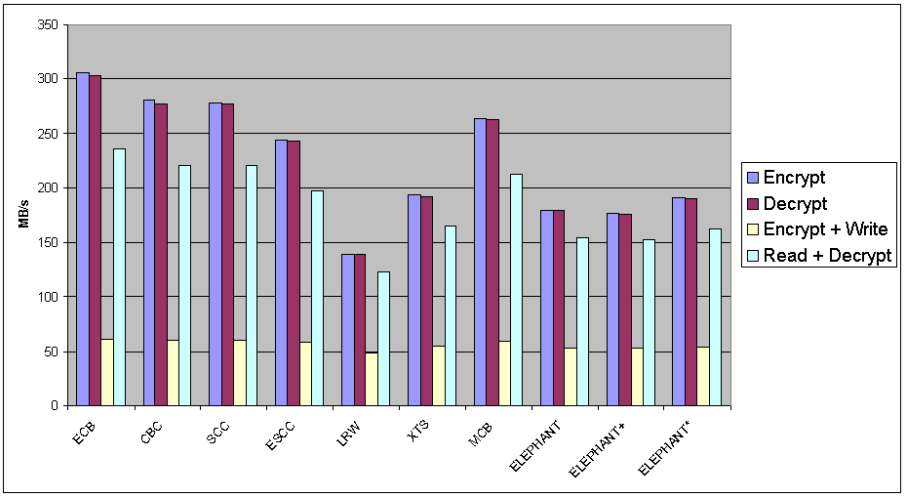


Figure 6.5: The average throughput of different modes of operation (Key=128, Step=1 MB, Loops=10).

The results of the benchmark for large data (Step=1 MB and Key=256) are shown in Figure 6.6. From these results, it is shown that SCC is almost as fast as CBC and ESCC is about 10% slower than CBC. MCB possesses high throughput, as it is about 40% faster than LRW and 23% faster than XTS. ELEPHANT$^+$ is as fast as ELEPAHNT, while ELEPHANT* is faster than ELEPHANT by about 5%.

6.8 Cryptanalysis of SCC

Let B_i^x be the ith block in the sector number x. The active tweak of the block B_{i+1}^x is defined by:

$$\tau_{i+1}^x = T_i^x \oplus B_i^x \tag{6.13}$$

By controlling the values of B_i^x, the values of τ_{i+1}^x can be controlled. Using the ability to modify τ_{i+1}^x, many attacks can be mounted.

6.8.1 Cryptanalysis of SCC-128

SCC-128 has the following set of unknown keys/masks: $\Pi = \{\kappa_i, \alpha_j, \beta_j, \tau^k : 0 \leq i \leq 10, i \neq x, y, z \text{ and } 0 \leq j \leq 31\}$, where κ_i is the round key of the i^{th} round and τ^k is the tweak of sector k.

The encryption key is recovered using three different attacks: **Attack1**, **Attack1′** and **Attack1″**, where **Attack1** and **Attack1′** use the active attack model in Sect. 6.1.5, while **Attack1″** uses the less restrictive attack model in Sect. 6.1.5.

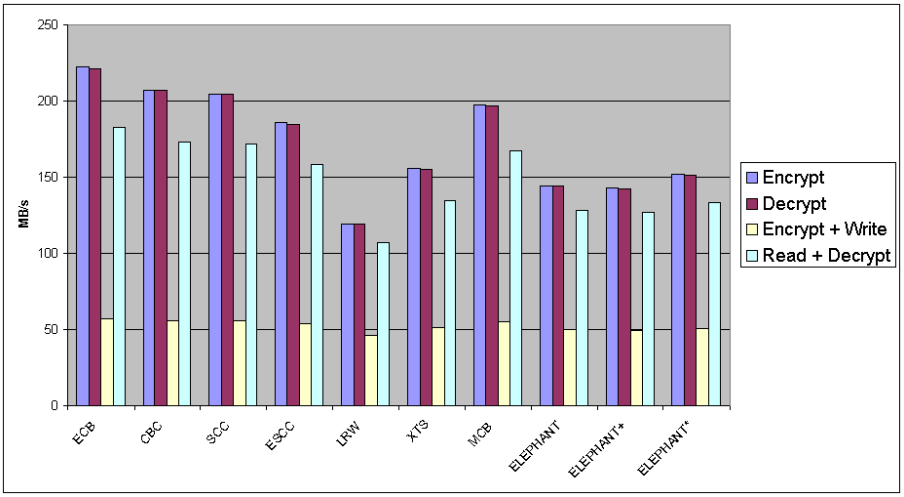


Figure 6.6: The average throughput of different modes of operation (Key=256, Step=1 MB, Loops=10).

Attack1

This attack is used to recover AES' round keys ($\{\kappa_i : 0 \leq i \leq 10, i \neq x, y, z\}$). Using the ability to modify τ^x_{i+1}, the square attack on the last 5 rounds of AES in SCC-128 can be mounted. The following procedure is used to generate Λ^1-sets:

1. Generate a set A of 256 plaintexts, so as A is a Λ^1-set.
2. For each plaintext $\mathrm{A}_i \in A$
   (a) Set $\mathrm{B}^x_i = \mathrm{A}_i$.
   (b) Read the sector x and ask the disk encryption application to re-encrypt it.
   (c) Store C^x_{i+1}, where C^x_{i+1} is the ciphertext of B^x_{i+1}.

This will create a Λ^1-set as the input to the 6^{th} round, which implies that the Square-5 attack can be applied on the rest 5 rounds to recover the round key of the last round, from which the rest of the round keys can be calculated. The cost of this attack is 5×2^8 chosen plaintext and requires about 2^{40} cipher executions (sector encryption operation) and this will reduce the unknown set to: $\Pi = \{\alpha_j, \beta_j, \tau^k : 0 \leq j \leq 31\}$, where τ^k is the tweak of sector k.

Attack1′

This attack is used to recover AES' round keys ($\{\kappa_i : 0 \leq i \leq 10, i \neq x, y, z\}$). Using the ability to modify τ^x_{i+1} the Pushdown-Square-5* attack (Sect. 3.3.3) can be mounted on the last 5 rounds of AES in SCC-128. The following procedure is used to generate a Λ^1-set:

1. Generate a set A of 256 plaintexts, so as A is an Ω^1-set.
2. For each plaintext $\mathrm{A}_i \in A$

(a) Set $B_i^x = A_i$.

(b) Read the sector x and ask the disk encryption application to re-encrypt it.

(c) Store C_{i+1}^x, where C_{i+1}^x is the ciphertext of B_{i+1}^x.

This will create an Ω^1-set as the input to the 6^{th}, which implies that the Pushdown-Square-5* attack (refer to Sect. 3.3.3) can be applied on the last 5 rounds, and obtaining the round key of the last round, from which the rest of the round keys can be calculated. The cost of this attack is 2^9 chosen plaintext and requires about 2^9 cipher executions (sector encryption operation) and this will reduce the unknown set to: $\Pi = \{\alpha_j, \beta_j, \tau^k : 0 \leq j \leq 31\}$, where τ^k is the tweak of sector k.

This attack is faster than **Attack1**.

Attack1″

This attack is used to recover AES' round keys ($\{\kappa_i : 0 \leq i \leq 10, i \neq x, y, z\}$). Using the ability to modify τ_{i+1}^x, the square attack on the first 6 rounds of AES can be mounted. The following procedure is used to generate Λ^1-sets:

1. Generate a set A of 256 plaintexts, so as A is a Λ^1-set.
2. For each plaintext $A_i \in A$
   (a) Set $B_i^x = A_i$.
   (b) Read the plaintext of sector x.
   (c) Store P_{i+1}^x, where P_{i+1}^x is the plaintext of B_{i+1}^x.

This will create a Λ^1-set as the input to the 6^{th} decryption round, which implies that the Square-5 attack can be applied on the first 5 decryption rounds and recovering the round key of the first round, from which the rest of the round keys can be calculated. The cost of this attack is 5×2^8 chosen plaintext and requires about 2^{40} cipher executions (sector encryption operation) and this will reduce the unknown set to: $\Pi = \{\alpha_j, \beta_j, \tau^k : 0 \leq j \leq 31\}$, where τ^k is the tweak of sector k.

Note that the square property, on which the square attack is built, is independent of the specific choices of SB, MC and the key schedule [38], thus the property holds in the decryption direction.

Attack2

This attack is used to recover β mask ($\{\beta_i : 0 \leq i \leq 31\}$).
Generate a set A of 256 plaintexts, so as A is a Λ^1-set. Apply two decryption AES round functions using the recovered κ_2 and κ_1, then xor the result with κ_0. Name the resulting set D. Note that encrypting the set D will result in a Λ^1-set as the input for the third round.

The attack works as follows, for each block B_i^x in a sector x:

1. Use D as its input and encrypt that block.
2. Decrypt the ciphertext till the 7^{th} round with the recovered round keys, name this set E.

3. Apply Square attack with L-representation (refer to Sect. 3.1.6), where the set A is the plaintext and the set E is the ciphertext.

4. The recovered round key is β_i.

The cost of this attack is 2^9 chosen plaintext and requires about 2^9 cipher executions, where τ^k is the tweak of sector k.

Attack3

This attack is used to recover τ^x.
Generate a set A of 256 plaintexts, so as A is a Λ^1-set. Apply one decryption AES round function using the recovered κ_1, then xor the result with κ_0. Name the resulting set C. Note that encrypting the set C will result in a Λ^1-set as the input for the second round.
The attack works as follows, for any block B_i^x in a sector x:

1. Use C as its input and encrypt that block.

2. Decrypt the ciphertext till the 8^{th} round with the known round keys, name this set D.

3. Apply Square attack with L-representation (refer to Sect. 3.1.6), where the set A is the plaintext and the set D is the ciphertext.

4. The recovered round key is τ^x.

The cost of this attack is 2^9 chosen plaintext and requires about 2^9 cipher executions. This will reduce the unknown set to: $\Pi = \{\alpha_j, \tau^k : 0 \leq j \leq 31\}$, where τ^k is the tweak of sector k and k$\neq$ x.

Attack4

This attack is used to recover α mask ($\{\alpha_i : 0 \leq i \leq 31\}$).
For the sector x, the set of unknowns is: $\Pi = \{\alpha_j : 0 \leq j \leq 31\}$, which can be recovered directly, by the following procedure:

- For each block B_i^x:

    1. Encrypt the plaintext X_i using the first three rounds and the known round keys to get Y_i.
    2. Apply SB, SR and MC on Y_i to produce W_i.
    3. Encrypt X_i using SCC to get A_i.
    4. Decrypt A_i using six rounds and the recovers round keys, sector ID and masks to get D_i.
    5. $\alpha_i = D_i \oplus W_i$

Now all the unknowns of the sector x are recovered, this attack requires only 1 known sector plaintext and ciphertext and about 1 cipher execution, this will reduce the unknown set to: $\Pi = \{\tau^k\}$, where τ^k is the tweak of sector k and k$\neq$ x.

Attack5

To recover the encrypted sector ID for a sector, the following procedure is used:

1. Encrypt a known plaintext X_i to get A_i.
2. Encrypt X_i using four rounds, the known round keys and α_i to get Y_i.
3. Apply SB, SR and MC on Y_i to produce W_i.
4. Decrypt A_i using five rounds with the known round keys and β_i to get D_i.
5. Calculate $\mathrm{TT}_i^x = D_i \oplus W_i$.
6. Calculate $\tau^x = TT_i^x \oplus B_{i-1}^x : B_{-1}^x = 0$.

This attack requires 1 known plaintext/ciphertext block and about 1 cipher execution.

Now, all the unknowns used by SCC-128 are recovered.

6.8.2 Cryptanalysis of SCC-256

AttackA

This attack is used to recover AES' round keys ($\{\kappa_i : 0 \leq i \leq 14, i \neq x, y, z\}$). Using the ability to modify τ_{i+1}^x Pushdown-Partial-Sum-7* attack (Sect. 3.3.4) can be mounted on the first 7 rounds of AES in SCC-256.
The following procedure is used to generate a Γ^4-set:

1. Generate a set A of 2^{32} plaintexts, so as A is a Γ^4-set.
2. For each plaintext $\mathrm{A}_i \in A$
   (a) Set $\mathrm{B}_i^x = \mathrm{A}_i$.
   (b) Read the plaintext of sector x.
   (c) Store P_{i+1}^x, where P_{i+1}^x is the plaintext of B_{i+1}^x.

This will create a Γ^4-set as the input to the 8^{th} decryption round, which implies that the Pushdown-Partial-Sum-7* attack (Sect. 3.3.4) can be applied on the last 7 decryption rounds, and obtaining the round key of the first round, from which the rest of the round keys can be calculated. The cost of this attack is 6×2^{32} chosen plaintext and requires about 2^{44} cipher executions (sector encryption operation). This will reduce the unknown set to: $\Pi = \{\alpha_j, \beta_j, \tau^k : 0 \leq j \leq 31\}$, where τ^k is the tweak of sector k.

Note that the square property on which the square attack is built on, is independent of the specific choices of SB, the multiplication polynomial of MC and the key schedule [38], thus the property holds in the decryption direction.

AttackB

This attack is used to recover 31 elements of α mask ($\{\alpha_i : 1 \leq i \leq 31\}$).
Generate a set A of 256 plaintexts, so as A is a Λ^1-set. Apply one decryption AES round function using the recovered κ_8. Name the resulting set D.

The attack works as follows, for each block B_i^x in a sector x, where $1 \leq i \leq 31$:

1. For each plaintext $D_i \in D$
   (a) Set $B_i^x = D_i$.
   (b) Read the plaintext of sector x.
   (c) Set $E_i = P_{i+1}^x$, where P_{i+1}^x is the plaintext of B_{i+1}^x.
   (d) Encrypt E_i using the first four encryption rounds with the recovered round keys, name the result F_i.
2. Apply Square attack with L-representation (refer to Sect. 3.1.6), where the set A is the plaintext and the set F is the ciphertext.
3. The recovered round key is α_i.

The cost of this attack is about 2^{10} chosen plaintext and requires about 2^{10} cipher executions (as $B_i^x : 1 \leq i \leq 31$ and i mod 2 =0 blocks can be executed in parallel, the same is valid for $B_i^x : 1 \leq i \leq 31$ and i mod 2 =1). This will reduce the unknown set to: $\Pi = \{\alpha_0, \beta_j, \tau^k : 0 \leq j \leq 31\}$, where τ^k is the tweak of sector k.

AttackC

This attack is used to recover 31 elements of β mask ($\{\beta_i : 1 \leq i \leq 31\}$).
The attack works as follows, for each block B_i^x in a sector x, where $1 \leq i \leq 31$:

1. Generate a set A of 256 plaintexts, so as A is a Λ^1-set.
2. For each plaintext $A_i \in A$
   (a) Set $B_i^x = A_i$.
   (b) Read the sector x and ask the disk encryption application to re-encrypt it.
   (c) Set C_i^x to the ciphertext of B_{i+1}^x.
3. Apply the first three decryption rounds on the set C with the recovered round keys, name this set D.
4. Apply Square attack with L-representation (refer to Sect. 3.1.6), where the set A is the plaintext and the set D is the ciphertext.
5. The recovered round key is β_i.

The cost of this attack is about 2^{10} chosen plaintext and requires about 2^{10} cipher executions (as $B_i^x : 1 \leq i \leq 31$ and i mod 2 =0 blocks can be executed in parallel, the same is valid for $B_i^x : 1 \leq i \leq 31$ and i mod 2 =1). This will reduce the unknown set to: $\Pi = \{\alpha_0, \beta_0, \tau^k\}$, where τ^k is the tweak of sector k.

AttackD

To recover the encrypted sector ID for sector x, the following procedure is used (where $i \neq 0$):

1. Encrypt a known plaintext X_i to get A_i.

2. Encrypt X_i using six rounds with the recovered round keys and α_i to get Y_i.
3. Apply SB, SR and MC on Y_i to produce W_i.
4. Decrypt A_i using seven rounds with the recovered round keys and β_i to get E_i.
5. Calculate $TT_i^x = E_i \oplus W_i$
6. Calculate $\tau^x = TT_i^x \oplus B_{i-1}^x$

This attack requires 1 known plaintext/ciphertext block and about 1 cipher execution. This will reduce the unknown set for sector x to: $\Pi = \{\alpha_0, \beta_0\}$.

AttackE

This attack is used to recover β_0 mask, the attack works as follows, for block B_0^x:

1. Generate a set A of 256 plaintexts, so that A is an Ω^1-set.
2. Decrypt the set A using the last 4 decryption rounds with the recovered round keys, name the result set C.
3. Apply SB, SR and MC on C to produce D.
4. Encrypt the set A and name the result set Y.
5. Decrypt set Y using the first four decryption rounds with the recovered round keys, name the result set W.
6. Apply the Pushdown-square-5* attack (Sect. 3.3.3) by replacing the square attack with the square attack with L-representation (refer to Sect. 3.1.6), where the set D is the plaintext and the set W is the ciphertext.
7. The recovered round key is β_0.

The cost of this attack is 2^9 chosen plaintext and requires about 2^9 cipher executions. This will reduce the unknown set for sector x to: $\Pi = \{\alpha_0\}$.

AttackF

This attack is used to recover α_0 mask.

1. Encrypt a known plaintext X_0 to get A_0.
2. Encrypt X_0 using four rounds with the recovered round keys to get Y_0.
3. Apply SB, SR and MC on Y_0 to produce W_0.
4. Decrypt A_i using nine rounds with the recovered round keys and β_0 to get C_0.
5. $\alpha_0 = C_0 \oplus W_0$

This attack requires 1 known plaintext/ciphertext block and about 1 cipher execution. This will reduce the unknown set to: $\Pi = \{\tau^k\}$, where τ^k is the tweak of sector k.

AttackG

To recover the encrypted sector ID for a sector, the following procedure is used:

1. Encrypt a known plaintext X_i to get A_i.
2. Encrypt X_i using six rounds with the recovered round keys and α_i to get Y_i.
3. Apply SB, SR and MC on Y_i to produce W_i.
4. Decrypt A_i using seven rounds with the recovered round keys and β_i to get C_i.
5. Calculate $TT_i^x = C_i \oplus W_i$
6. Calculate $\tau^x = TT_i^x \oplus B_{i-1}^x : B_{-1}^x = 0$

This attack requires 1 known plaintext/ciphertext block and about 1 cipher execution.

Now, all the unknowns used by SCC-256 are recovered.

6.8.3 Attacking ELEPHANT$^+$ and ELEPHANT$^\times$

The attacks presented in this section cannot be applied on ELEPHANT$^+$ and ELEPHANT$^\times$. The reason is that, if some bits of the ciphertext are changed, the decryption operation will be associated with the avalanche effect [117], due to the existence of the diffusers. Thus these attacks cannot be applied.

6.9 Summary

In this chapter, three novel modes of operation for disk encryption applications are proposed. The first mode is a secure narrow-block mode of operation that operates in parallel: this mode of operation is faster than the current IEEE standard and does not face its limitations. The second mode is used to modify Windows Vista's encryption algorithm to enhance some of its diffusion properties and offers some parallelization to its original design. This mode is not designed to be used as a stand alone mode of operation, as it manipulates the cipher's inner state. To emphasize on this point, this mode of operation is cryptanalyzed and is proven to be broken if used as a stand alone mode of operation. The third mode is a secure narrow-block mode of operation that offers error propagation.

Chapter 7

Conclusions and Outlook

This thesis develops several cryptographic techniques that can enhance the current approaches. These techniques are applied in real applications. In the following, the major contributions of this thesis are recalled:

1. A new idea on chosen plaintext cryptanalysis is presented where some of the cipher's encryption rounds at its beginning can be bypassed. To illustrate this idea, the Pushdown attacks are developed. These attacks can increase the effectiveness of some chosen plaintext attacks. The Pushdown attacks are applied on AES and are able to achieve a 6-round attack that requires only 2^{11} chosen plaintexts. This reduces the chosen plaintexts needed by the Square attack by a factor of 2^{21}. This idea is also extended and applied on chosen ciphertext, chosen plaintext-adaptive chosen ciphertext and chosen ciphertext-adaptive chosen plaintext attacks. Furthermore, some of the proposed attacks are used to break the Substitution Cipher Chaining mode (SCC) for disk encryption.

2. An enhanced key schedule for AES is proposed. This key schedule raises the classification of AES' key schedule from **1C** to **2B**. In addition, it protects the current AES implementation from many attacks like related-key and some cache timing attacks. It is worth mentioning that this proposal increases the time complexity of many attacks, even the exhaustive key search. A generalized block ciphers' key schedule is also presented.

3. Three novel encryption models are proposed:

   Dynamic Substitution Model (DSM) is a model that can provide a block cipher with a *variable* length secondary key. The secondary key is used to replace some words of the cipher's expanded key. **Static Substitution Model (SSM)**, a special case of DSM that uses *fixed* length secondary key, is also developed.

   Dynamic Injection Model (DIM) is a model that can provide a block cipher with *variable* length secondary key. The secondary key is consumed by some keyed transformation functions that are injected into the cipher. **Static Injection Model (SIM)**, a special case of DIM that uses *fixed* length secondary key, is also developed.

Dynamic Permutation Model (DPM) is a model that can provide a block cipher with a *fixed* length secondary key. The secondary key is used to permute some words of the cipher's expanded key.

These models can be efficiently used as building blocks for new encryption modes of operation. They are characterized by their low memory requirements and high speed. The guidelines to securely use these models are presented together with the possibility of constructing hybrid models, using these three models as their building blocks.

4. Using the proposed encryption models, new variants of AES are proposed. These variants allow AES to accept secondary key(s). With the help of these variants, new efficient network encryption schemes are proposed. These schemes are characterized by their high throughput, stability, low setup time, low memory requirements, scalability and security. The three proposed network encryption schemes are:

   CBC-S/CTR-S are network encryption schemes based on CBC/CTR. CBC-S/CTR-S require much less memory resources than the classical CBC/CTR schemes. Thus, they increase the number of concurrent clients that a server can serve. CBC-S/CTR-S can save up to 94% of the memory required by the classical CBC/CTR schemes. The experimental results demonstrated that CBC-S/CTR-S are faster than the classical CBC/CTR schemes.

   GSCM is a set of network encryption schemes based on GCM. GSCM schemes require much less memory resources than the classical GCM schemes. They increase the number of concurrent clients a server can serve. GSCM can save up to more than 99% of the memory required by some classical GCM schemes. The experimental results demonstrated that GSCM schemes are faster than the classical GCM schemes.

5. The new proposed variants of AES are used to design new modes of operation dedicated to disk encryption applications. These modes of operation are characterized by their high throughput and their resistance to some known attacks, when compared to the state of the art modes of operation. The three proposed modes of operation for disk encryption are:

   Masked Code Book (MCB) is designed to serve as a narrow-block mode of operation that can be parallelized. It is about 25% faster than the current IEEE standard (XTS) and additionally resistant to the attacks on XTS.

   Substitution Cipher Chaining mode (SCC) is designed to modify Windows Vista's Disk Encryption algorithm "ELEPHANT", and it is not designed to be used as a stand alone disk encryption mode of operation. SCC is used to design two variants of ELEPHANT, which are faster and possess higher Safety Factors than ELEPHANT.

   Extended Substitution Cipher Chaining mode (ESCC) is designed to serve as a narrow-block mode of operation that offers error propagation. It is about 8% slower than CBC but more resistant to manipulation attacks.

In the course of the thesis, the author found several issues that can extend the work in this thesis and present interesting directions for future research:

- Applying the proposed models to block ciphers other than AES.
- Applying the Pushup and Sandwich attacks to AES.
- Applying the Pushdown, Pushup and Sandwich attacks to block ciphers other than AES.
- Constructing more hybrid encryption models, using the proposed models.
- Studying the effect of the proposed key scheduling on side channel attacks
- Developing more security applications based on the proposed models.

Appendix A

MBOX Values

The values of MBOX used in DP-AES (refer to Sect. 4.8.1).

```
MBOX[256][8]={
{ 5, 4, 7, 2, 3, 0, 6, 1},
{ 6, 5, 0, 3, 4, 1, 7, 2},
{ 7, 6, 1, 4, 5, 2, 0, 3},
{ 0, 7, 2, 5, 6, 3, 1, 4},
{ 1, 0, 3, 6, 7, 4, 2, 5},
{ 2, 1, 4, 7, 0, 5, 3, 6},
{ 3, 2, 5, 0, 1, 6, 4, 7},
{ 4, 3, 6, 1, 2, 7, 5, 0},
{ 5, 3, 1, 4, 0, 2, 6, 7},
{ 6, 4, 2, 5, 1, 3, 7, 0},
{ 7, 5, 3, 6, 2, 4, 0, 1},
{ 0, 6, 4, 7, 3, 5, 1, 2},
{ 1, 7, 5, 0, 4, 6, 2, 3},
{ 2, 0, 6, 1, 5, 7, 3, 4},
{ 3, 1, 7, 2, 6, 0, 4, 5},
{ 4, 2, 0, 3, 7, 1, 5, 6},
{ 0, 5, 1, 7, 6, 4, 3, 2},
{ 1, 6, 2, 0, 7, 5, 4, 3},
{ 2, 7, 3, 1, 0, 6, 5, 4},
{ 3, 0, 4, 2, 1, 7, 6, 5},
{ 4, 1, 5, 3, 2, 0, 7, 6},
{ 5, 2, 6, 4, 3, 1, 0, 7},
{ 6, 3, 7, 5, 4, 2, 1, 0},
{ 7, 4, 0, 6, 5, 3, 2, 1},
{ 1, 5, 0, 2, 4, 3, 7, 6},
{ 2, 6, 1, 3, 5, 4, 0, 7},
{ 3, 7, 2, 4, 6, 5, 1, 0},
{ 4, 0, 3, 5, 7, 6, 2, 1},
{ 5, 1, 4, 6, 0, 7, 3, 2},
{ 6, 2, 5, 7, 1, 0, 4, 3},
{ 7, 3, 6, 0, 2, 1, 5, 4},
{ 0, 4, 7, 1, 3, 2, 6, 5},
{ 4, 0, 7, 5, 3, 1, 6, 2},
{ 5, 1, 0, 6, 4, 2, 7, 3},
{ 6, 2, 1, 7, 5, 3, 0, 4},
{ 7, 3, 2, 0, 6, 4, 1, 5},
```

```
{ 0, 4, 3, 1, 7, 5, 2, 6},
{ 1, 5, 4, 2, 0, 6, 3, 7},
{ 2, 6, 5, 3, 1, 7, 4, 0},
{ 3, 7, 6, 4, 2, 0, 5, 1},
{ 0, 5, 7, 2, 6, 4, 3, 1},
{ 1, 6, 0, 3, 7, 5, 4, 2},
{ 2, 7, 1, 4, 0, 6, 5, 3},
{ 3, 0, 2, 5, 1, 7, 6, 4},
{ 4, 1, 3, 6, 2, 0, 7, 5},
{ 5, 2, 4, 7, 3, 1, 0, 6},
{ 6, 3, 5, 0, 4, 2, 1, 7},
{ 7, 4, 6, 1, 5, 3, 2, 0},
{ 3, 1, 7, 5, 2, 0, 4, 6},
{ 4, 2, 0, 6, 3, 1, 5, 7},
{ 5, 3, 1, 7, 4, 2, 6, 0},
{ 6, 4, 2, 0, 5, 3, 7, 1},
{ 7, 5, 3, 1, 6, 4, 0, 2},
{ 0, 6, 4, 2, 7, 5, 1, 3},
{ 1, 7, 5, 3, 0, 6, 2, 4},
{ 2, 0, 6, 4, 1, 7, 3, 5},
{ 6, 2, 0, 5, 4, 1, 3, 7},
{ 7, 3, 1, 6, 5, 2, 4, 0},
{ 0, 4, 2, 7, 6, 3, 5, 1},
{ 1, 5, 3, 0, 7, 4, 6, 2},
{ 2, 6, 4, 1, 0, 5, 7, 3},
{ 3, 7, 5, 2, 1, 6, 0, 4},
{ 4, 0, 6, 3, 2, 7, 1, 5},
{ 5, 1, 7, 4, 3, 0, 2, 6},
{ 0, 5, 4, 7, 3, 2, 1, 6},
{ 1, 6, 5, 0, 4, 3, 2, 7},
{ 2, 7, 6, 1, 5, 4, 3, 0},
{ 3, 0, 7, 2, 6, 5, 4, 1},
{ 4, 1, 0, 3, 7, 6, 5, 2},
{ 5, 2, 1, 4, 0, 7, 6, 3},
{ 6, 3, 2, 5, 1, 0, 7, 4},
{ 7, 4, 3, 6, 2, 1, 0, 5},
{ 2, 7, 6, 3, 0, 5, 1, 4},
{ 3, 0, 7, 4, 1, 6, 2, 5},
{ 4, 1, 0, 5, 2, 7, 3, 6},
{ 5, 2, 1, 6, 3, 0, 4, 7},
{ 6, 3, 2, 7, 4, 1, 5, 0},
{ 7, 4, 3, 0, 5, 2, 6, 1},
{ 0, 5, 4, 1, 6, 3, 7, 2},
{ 1, 6, 5, 2, 7, 4, 0, 3},
{ 0, 5, 3, 1, 4, 7, 2, 6},
{ 1, 6, 4, 2, 5, 0, 3, 7},
{ 2, 7, 5, 3, 6, 1, 4, 0},
{ 3, 0, 6, 4, 7, 2, 5, 1},
{ 4, 1, 7, 5, 0, 3, 6, 2},
{ 5, 2, 0, 6, 1, 4, 7, 3},
{ 6, 3, 1, 7, 2, 5, 0, 4},
{ 7, 4, 2, 0, 3, 6, 1, 5},
{ 5, 3, 0, 1, 4, 7, 2, 6},
{ 6, 4, 1, 2, 5, 0, 3, 7},
```

```
{ 7, 5, 2, 3, 6, 1, 4, 0},
{ 0, 6, 3, 4, 7, 2, 5, 1},
{ 1, 7, 4, 5, 0, 3, 6, 2},
{ 2, 0, 5, 6, 1, 4, 7, 3},
{ 3, 1, 6, 7, 2, 5, 0, 4},
{ 4, 2, 7, 0, 3, 6, 1, 5},
{ 4, 6, 5, 2, 3, 1, 0, 7},
{ 5, 7, 6, 3, 4, 2, 1, 0},
{ 6, 0, 7, 4, 5, 3, 2, 1},
{ 7, 1, 0, 5, 6, 4, 3, 2},
{ 0, 2, 1, 6, 7, 5, 4, 3},
{ 1, 3, 2, 7, 0, 6, 5, 4},
{ 2, 4, 3, 0, 1, 7, 6, 5},
{ 3, 5, 4, 1, 2, 0, 7, 6},
{ 2, 6, 4, 1, 5, 0, 3, 7},
{ 3, 7, 5, 2, 6, 1, 4, 0},
{ 4, 0, 6, 3, 7, 2, 5, 1},
{ 5, 1, 7, 4, 0, 3, 6, 2},
{ 6, 2, 0, 5, 1, 4, 7, 3},
{ 7, 3, 1, 6, 2, 5, 0, 4},
{ 0, 4, 2, 7, 3, 6, 1, 5},
{ 1, 5, 3, 0, 4, 7, 2, 6},
{ 5, 4, 1, 3, 7, 6, 2, 0},
{ 6, 5, 2, 4, 0, 7, 3, 1},
{ 7, 6, 3, 5, 1, 0, 4, 2},
{ 0, 7, 4, 6, 2, 1, 5, 3},
{ 1, 0, 5, 7, 3, 2, 6, 4},
{ 2, 1, 6, 0, 4, 3, 7, 5},
{ 3, 2, 7, 1, 5, 4, 0, 6},
{ 4, 3, 0, 2, 6, 5, 1, 7},
{ 3, 1, 7, 2, 5, 0, 6, 4},
{ 4, 2, 0, 3, 6, 1, 7, 5},
{ 5, 3, 1, 4, 7, 2, 0, 6},
{ 6, 4, 2, 5, 0, 3, 1, 7},
{ 7, 5, 3, 6, 1, 4, 2, 0},
{ 0, 6, 4, 7, 2, 5, 3, 1},
{ 1, 7, 5, 0, 3, 6, 4, 2},
{ 2, 0, 6, 1, 4, 7, 5, 3},
{ 3, 1, 7, 0, 4, 6, 5, 2},
{ 4, 2, 0, 1, 5, 7, 6, 3},
{ 5, 3, 1, 2, 6, 0, 7, 4},
{ 6, 4, 2, 3, 7, 1, 0, 5},
{ 7, 5, 3, 4, 0, 2, 1, 6},
{ 0, 6, 4, 5, 1, 3, 2, 7},
{ 1, 7, 5, 6, 2, 4, 3, 0},
{ 2, 0, 6, 7, 3, 5, 4, 1},
{ 4, 6, 2, 1, 5, 0, 3, 7},
{ 5, 7, 3, 2, 6, 1, 4, 0},
{ 6, 0, 4, 3, 7, 2, 5, 1},
{ 7, 1, 5, 4, 0, 3, 6, 2},
{ 0, 2, 6, 5, 1, 4, 7, 3},
{ 1, 3, 7, 6, 2, 5, 0, 4},
{ 2, 4, 0, 7, 3, 6, 1, 5},
{ 3, 5, 1, 0, 4, 7, 2, 6},
```

```
{ 1, 6, 4, 7, 2, 0, 5, 3},
{ 2, 7, 5, 0, 3, 1, 6, 4},
{ 3, 0, 6, 1, 4, 2, 7, 5},
{ 4, 1, 7, 2, 5, 3, 0, 6},
{ 5, 2, 0, 3, 6, 4, 1, 7},
{ 6, 3, 1, 4, 7, 5, 2, 0},
{ 7, 4, 2, 5, 0, 6, 3, 1},
{ 0, 5, 3, 6, 1, 7, 4, 2},
{ 2, 6, 1, 0, 5, 7, 4, 3},
{ 3, 7, 2, 1, 6, 0, 5, 4},
{ 4, 0, 3, 2, 7, 1, 6, 5},
{ 5, 1, 4, 3, 0, 2, 7, 6},
{ 6, 2, 5, 4, 1, 3, 0, 7},
{ 7, 3, 6, 5, 2, 4, 1, 0},
{ 0, 4, 7, 6, 3, 5, 2, 1},
{ 1, 5, 0, 7, 4, 6, 3, 2},
{ 1, 4, 3, 0, 2, 7, 5, 6},
{ 2, 5, 4, 1, 3, 0, 6, 7},
{ 3, 6, 5, 2, 4, 1, 7, 0},
{ 4, 7, 6, 3, 5, 2, 0, 1},
{ 5, 0, 7, 4, 6, 3, 1, 2},
{ 6, 1, 0, 5, 7, 4, 2, 3},
{ 7, 2, 1, 6, 0, 5, 3, 4},
{ 0, 3, 2, 7, 1, 6, 4, 5},
{ 0, 4, 6, 5, 2, 1, 3, 7},
{ 1, 5, 7, 6, 3, 2, 4, 0},
{ 2, 6, 0, 7, 4, 3, 5, 1},
{ 3, 7, 1, 0, 5, 4, 6, 2},
{ 4, 0, 2, 1, 6, 5, 7, 3},
{ 5, 1, 3, 2, 7, 6, 0, 4},
{ 6, 2, 4, 3, 0, 7, 1, 5},
{ 7, 3, 5, 4, 1, 0, 2, 6},
{ 1, 6, 2, 5, 7, 3, 4, 0},
{ 2, 7, 3, 6, 0, 4, 5, 1},
{ 3, 0, 4, 7, 1, 5, 6, 2},
{ 4, 1, 5, 0, 2, 6, 7, 3},
{ 5, 2, 6, 1, 3, 7, 0, 4},
{ 6, 3, 7, 2, 4, 0, 1, 5},
{ 7, 4, 0, 3, 5, 1, 2, 6},
{ 0, 5, 1, 4, 6, 2, 3, 7},
{ 4, 7, 1, 5, 6, 3, 2, 0},
{ 5, 0, 2, 6, 7, 4, 3, 1},
{ 6, 1, 3, 7, 0, 5, 4, 2},
{ 7, 2, 4, 0, 1, 6, 5, 3},
{ 0, 3, 5, 1, 2, 7, 6, 4},
{ 1, 4, 6, 2, 3, 0, 7, 5},
{ 2, 5, 7, 3, 4, 1, 0, 6},
{ 3, 6, 0, 4, 5, 2, 1, 7},
{ 2, 1, 7, 3, 5, 4, 0, 6},
{ 3, 2, 0, 4, 6, 5, 1, 7},
{ 4, 3, 1, 5, 7, 6, 2, 0},
{ 5, 4, 2, 6, 0, 7, 3, 1},
{ 6, 5, 3, 7, 1, 0, 4, 2},
{ 7, 6, 4, 0, 2, 1, 5, 3},
```

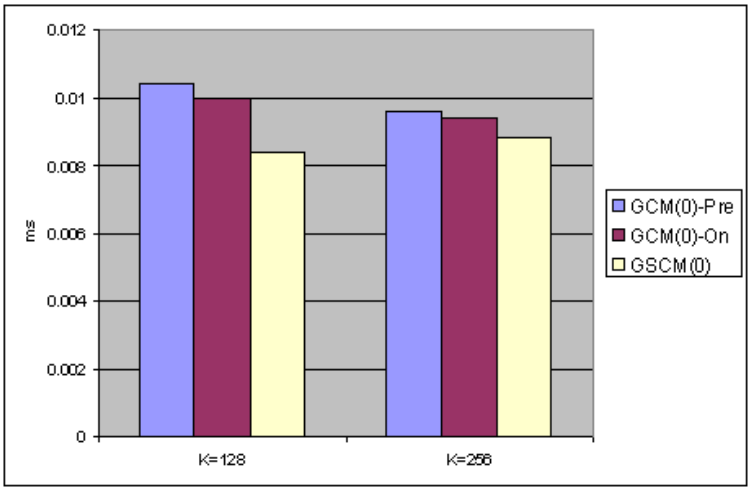
```
{ 0, 7, 5, 1, 3, 2, 6, 4},
{ 1, 0, 6, 2, 4, 3, 7, 5},
{ 1, 5, 0, 2, 7, 3, 6, 4},
{ 2, 6, 1, 3, 0, 4, 7, 5},
{ 3, 7, 2, 4, 1, 5, 0, 6},
{ 4, 0, 3, 5, 2, 6, 1, 7},
{ 5, 1, 4, 6, 3, 7, 2, 0},
{ 6, 2, 5, 7, 4, 0, 3, 1},
{ 7, 3, 6, 0, 5, 1, 4, 2},
{ 0, 4, 7, 1, 6, 2, 5, 3},
{ 4, 7, 3, 0, 6, 5, 1, 2},
{ 5, 0, 4, 1, 7, 6, 2, 3},
{ 6, 1, 5, 2, 0, 7, 3, 4},
{ 7, 2, 6, 3, 1, 0, 4, 5},
{ 0, 3, 7, 4, 2, 1, 5, 6},
{ 1, 4, 0, 5, 3, 2, 6, 7},
{ 2, 5, 1, 6, 4, 3, 7, 0},
{ 3, 6, 2, 7, 5, 4, 0, 1},
{ 2, 6, 3, 1, 4, 0, 5, 7},
{ 3, 7, 4, 2, 5, 1, 6, 0},
{ 4, 0, 5, 3, 6, 2, 7, 1},
{ 5, 1, 6, 4, 7, 3, 0, 2},
{ 6, 2, 7, 5, 0, 4, 1, 3},
{ 7, 3, 0, 6, 1, 5, 2, 4},
{ 0, 4, 1, 7, 2, 6, 3, 5},
{ 1, 5, 2, 0, 3, 7, 4, 6},
{ 4, 1, 6, 2, 5, 7, 3, 0},
{ 5, 2, 7, 3, 6, 0, 4, 1},
{ 6, 3, 0, 4, 7, 1, 5, 2},
{ 7, 4, 1, 5, 0, 2, 6, 3},
{ 0, 5, 2, 6, 1, 3, 7, 4},
{ 1, 6, 3, 7, 2, 4, 0, 5},
{ 2, 7, 4, 0, 3, 5, 1, 6},
{ 3, 0, 5, 1, 4, 6, 2, 7},
{ 6, 3, 0, 4, 5, 7, 1, 2},
{ 7, 4, 1, 5, 6, 0, 2, 3},
{ 0, 5, 2, 6, 7, 1, 3, 4},
{ 1, 6, 3, 7, 0, 2, 4, 5},
{ 2, 7, 4, 0, 1, 3, 5, 6},
{ 3, 0, 5, 1, 2, 4, 6, 7},
{ 4, 1, 6, 2, 3, 5, 7, 0},
{ 5, 2, 7, 3, 4, 6, 0, 1},
{ 3, 2, 6, 4, 1, 7, 0, 5},
{ 4, 3, 7, 5, 2, 0, 1, 6},
{ 5, 4, 0, 6, 3, 1, 2, 7},
{ 6, 5, 1, 7, 4, 2, 3, 0},
{ 7, 6, 2, 0, 5, 3, 4, 1},
{ 0, 7, 3, 1, 6, 4, 5, 2},
{ 1, 0, 4, 2, 7, 5, 6, 3},
{ 2, 1, 5, 3, 0, 6, 7, 4},
{ 1, 2, 4, 0, 6, 3, 5, 7},
{ 2, 3, 5, 1, 7, 4, 6, 0},
{ 3, 4, 6, 2, 0, 5, 7, 1},
{ 4, 5, 7, 3, 1, 6, 0, 2},
```

```
{ 5, 6, 0, 4, 2, 7, 1, 3},
{ 6, 7, 1, 5, 3, 0, 2, 4},
{ 7, 0, 2, 6, 4, 1, 3, 5},
{ 0, 1, 3, 7, 5, 2, 4, 6}
};
```

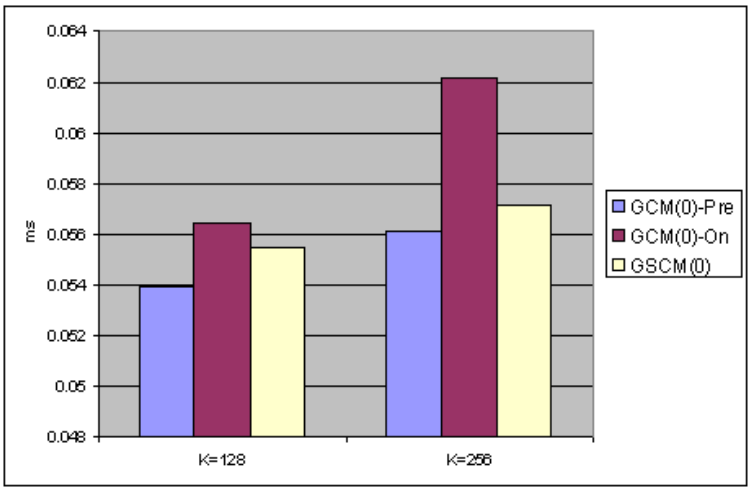
Appendix B

Performance Simulation Results

The results of the performance simulation for GCM(0), GCM(256), GCM(4k), GCM(8k) and GCM(64k) schemes are shown in Figure B.1, Figure B.2, Figure B.3, Figure B.4 and Figure B.5 respectively. These results demonstrate that GSCM(x) is faster than GCM(x)-Pre and GCM(x)-On.

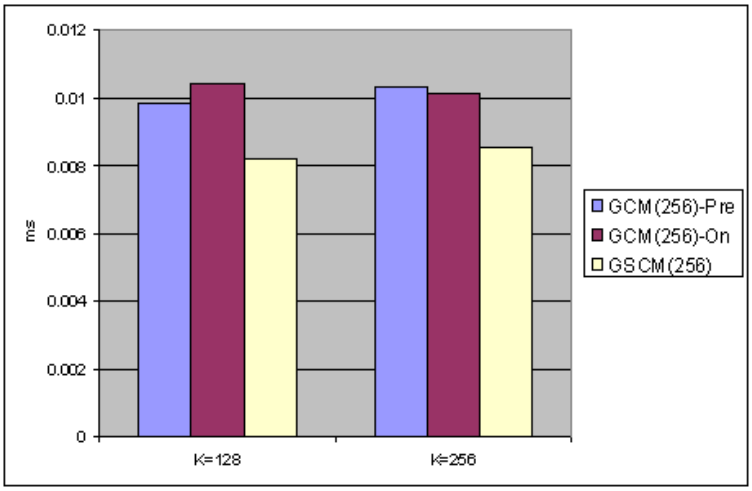


(a) Simulation parameters (α=40 and UNIT=10,000).

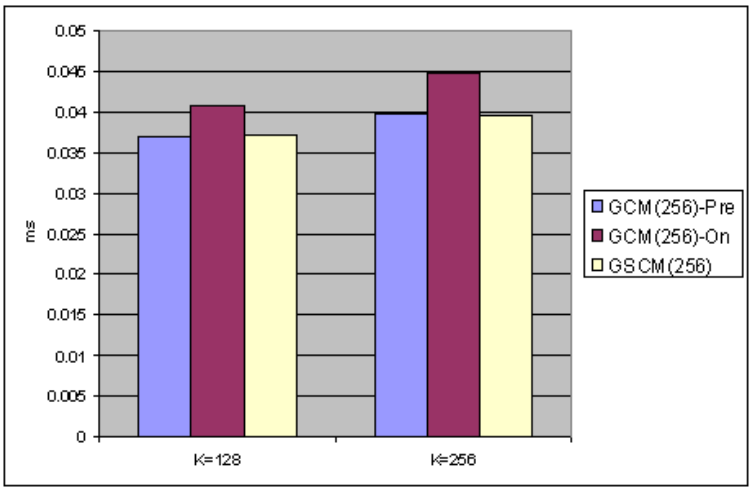


(b) Simulation parameters (α=1500 and UNIT=10,000).

Figure B.1: Performance analysis simulation: average time (in ms) needed to process a packet, using GCM(0) Schemes.

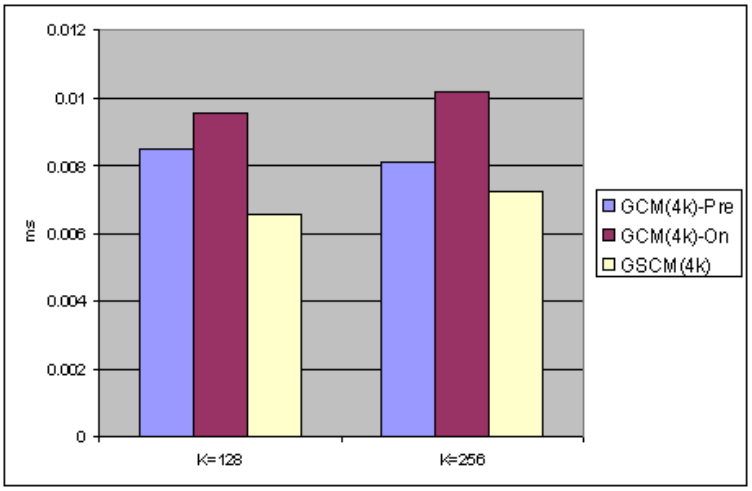


(a) Simulation parameters (α=40 and UNIT=10,000).

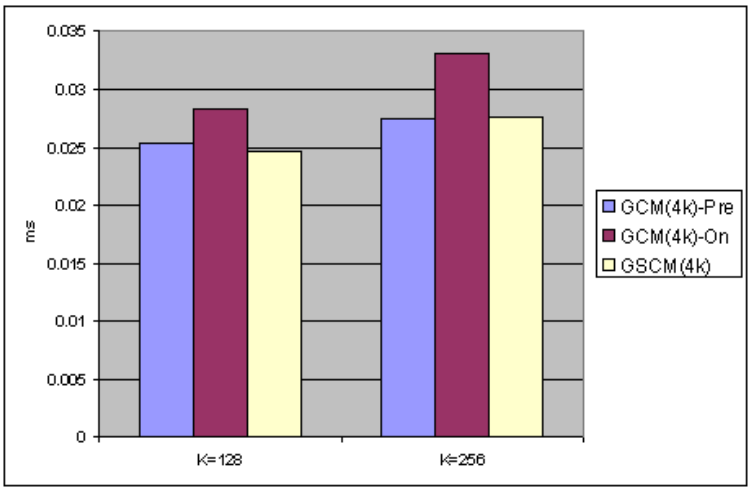


(b) Simulation parameters (α=1500 and UNIT=10,000).

Figure B.2: Performance analysis simulation: average time (in ms) needed to process a packet, using GCM(256) Schemes.

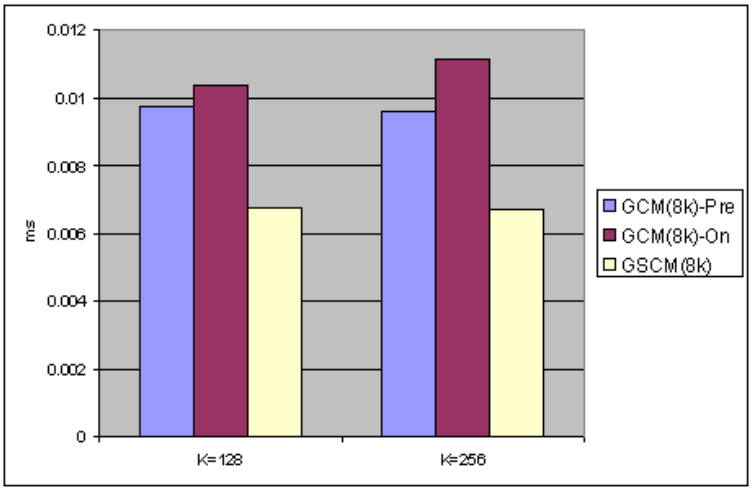


(a) Simulation parameters (α=40 and UNIT=10,000).

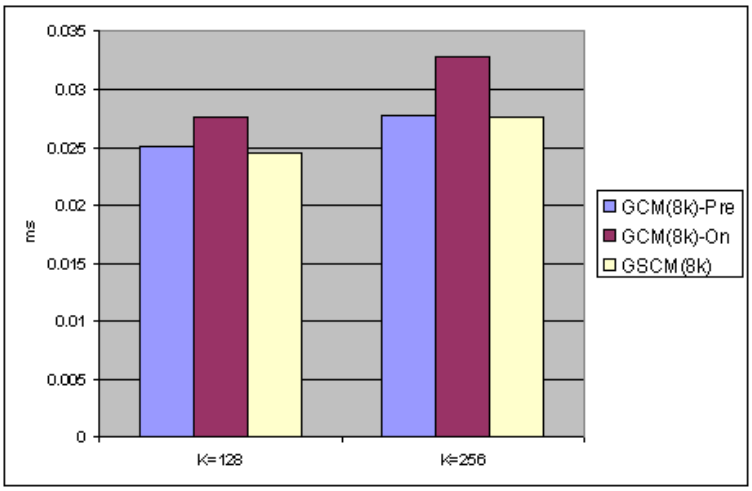


(b) Simulation parameters (α=1500 and UNIT=10,000).

Figure B.3: Performance analysis simulation: average time (in ms) needed to process a packet, using GCM(4k) Schemes.

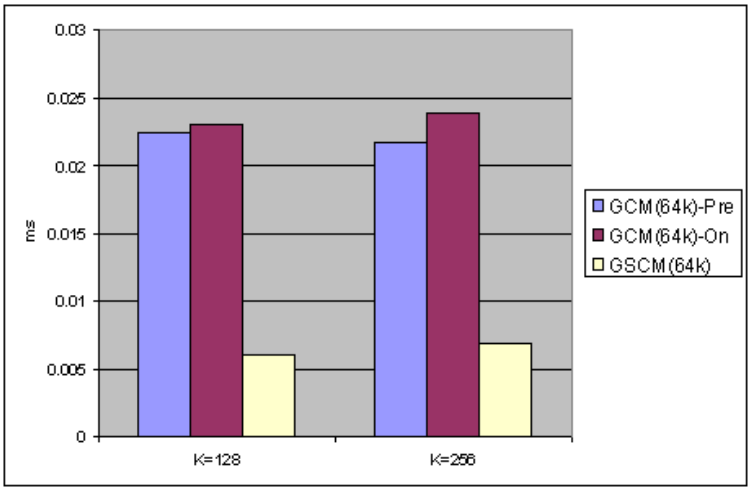


(a) Simulation parameters (α=40 and UNIT=10,000).

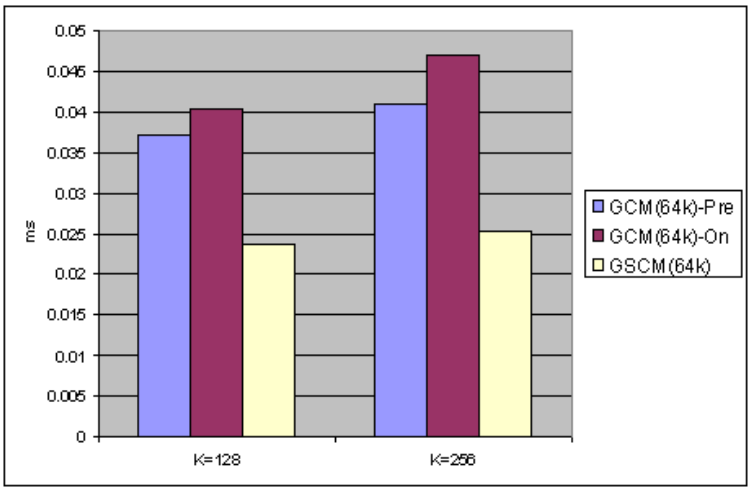


(b) Simulation parameters (α=1500 and UNIT=10,000).

Figure B.4: Performance analysis simulation: average time (in ms) needed to process a packet, using GCM(8k) Schemes.



(a) Simulation parameters (α=40 and UNIT=10,000).



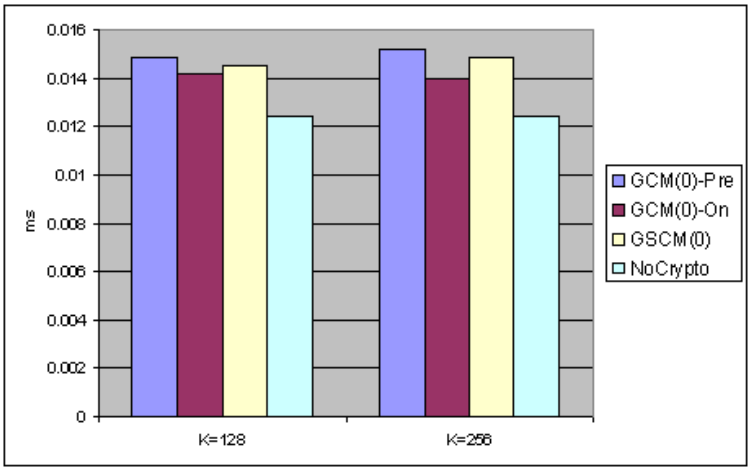
(b) Simulation parameters (α=1500 and UNIT=10,000).

Figure B.5: Performance analysis simulation: average time (in ms) needed to process a packet, using GCM(64k) Schemes.

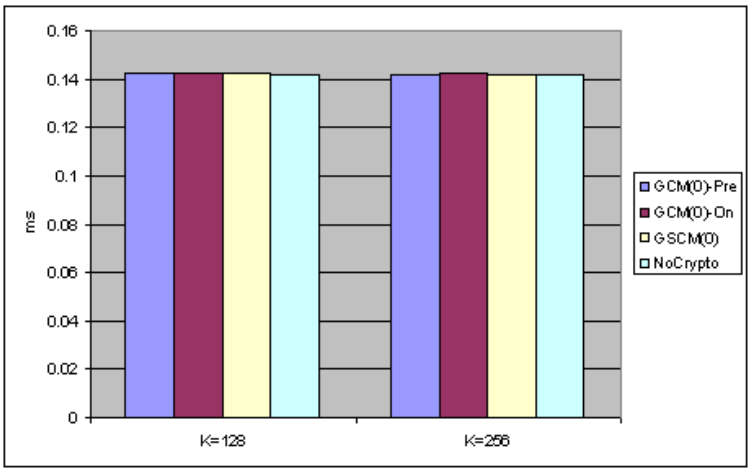
Appendix C

Network Simulation Results

Figure C.1, Figure C.2, Figure C.3, Figure C.4 and Figure C.5 present the simulation results for GCM(0), GCM(256), GCM(4k), GCM(8k) and GCM(64k) schemes, respectively. From these results, it is clear that GSCM(x) schemes are faster than GCM(x)-Pre and GCM(x)-On schemes in encrypting small packets. For large packets, the encryption overhead is considered negligible for all the schemes.

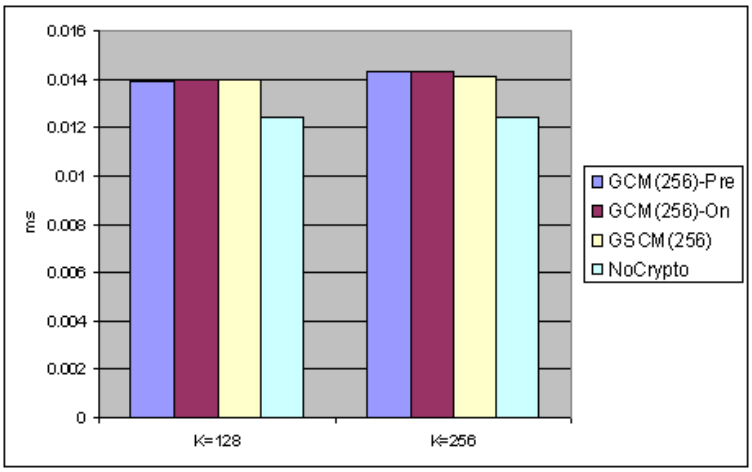


(a) Simulation parameters (α=40 and UNIT=100,000).

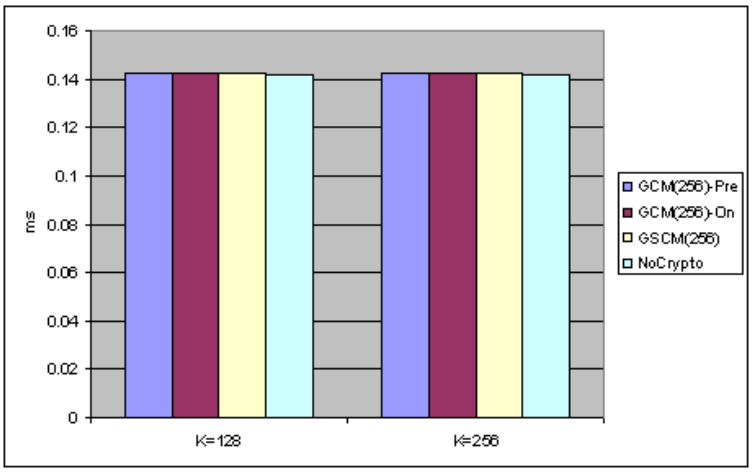


(b) Simulation parameters (α=1500 and UNIT=100,000).

Figure C.1: Network simulation: average time (in ms) needed to process a packet, using GCM(0) Schemes.

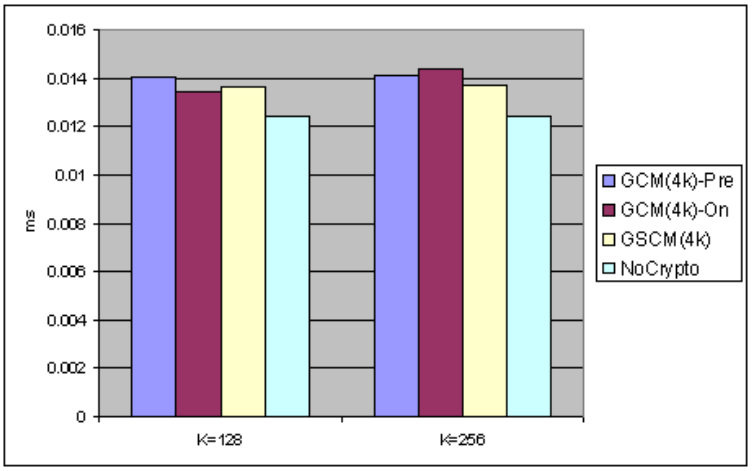


(a) Simulation parameters (α=40 and UNIT=10,000).

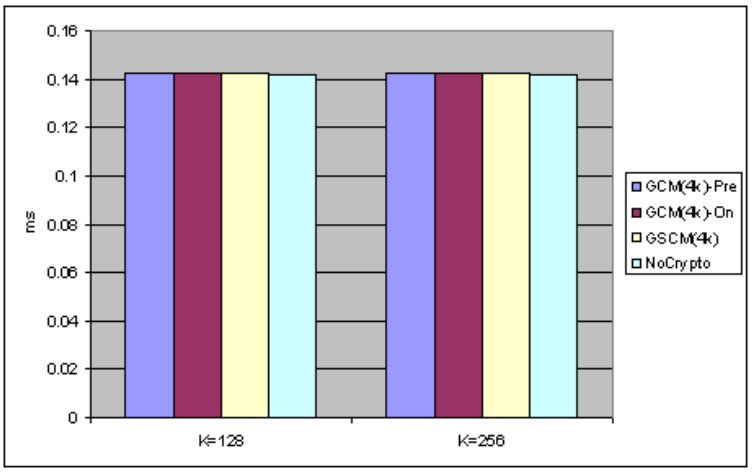


(b) Simulation parameters (α=1500 and UNIT=10,000).

Figure C.2: Network simulation: average time (in ms) needed to process a packet, using GCM(256) Schemes.

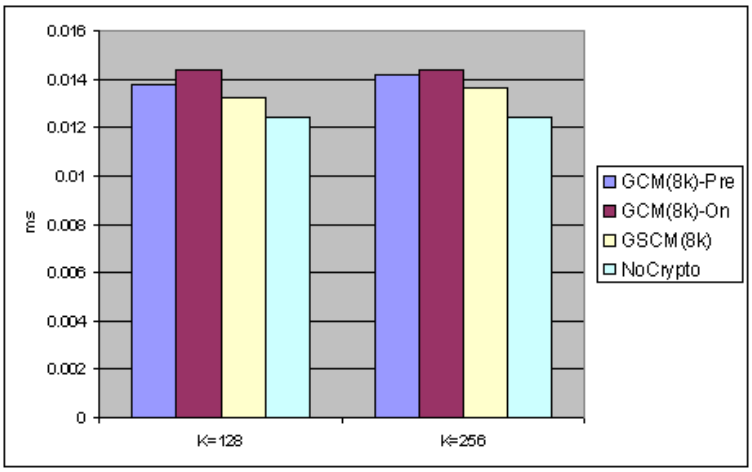


(a) Simulation parameters (α=40 and UNIT=10,000).

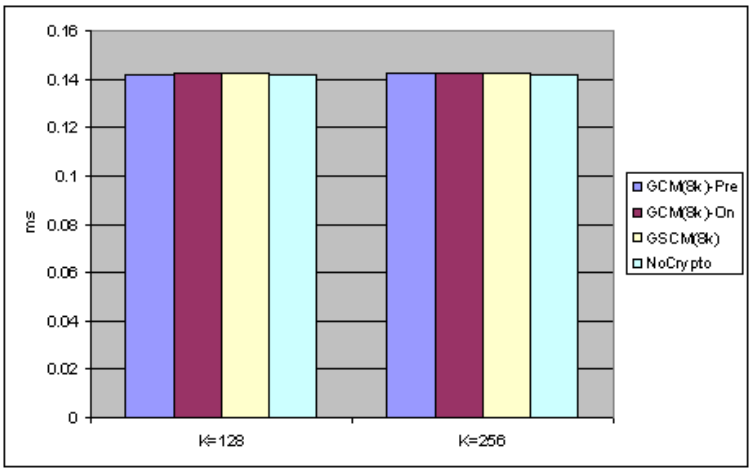


(b) Simulation parameters (α=1500 and UNIT=10,000).

Figure C.3: Network simulation: average time (in ms) needed to process a packet, using GCM(4k) Schemes.

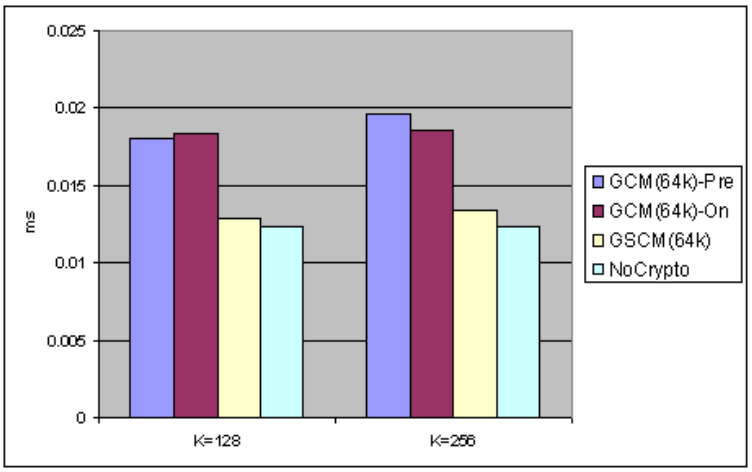


(a) Simulation parameters (α=40 and UNIT=10,000).

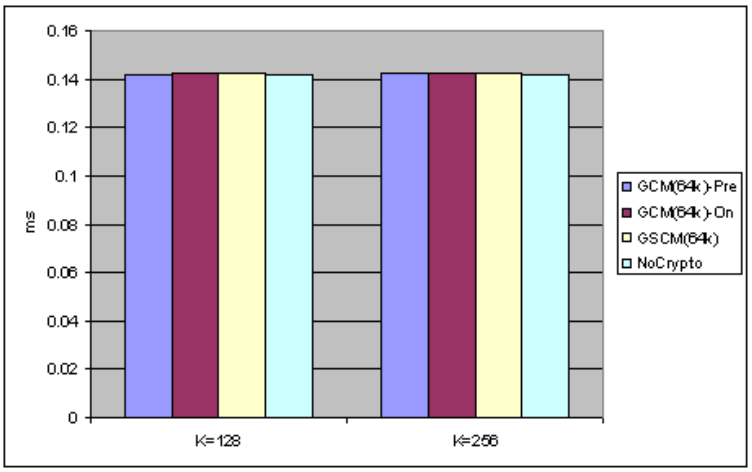


(b) Simulation parameters (α=1500 and UNIT=10,000).

Figure C.4: Network simulation: average time (in ms) needed to process a packet, using GCM(8k) Schemes.



(a) Simulation parameters (α=40 and UNIT=10,000).



(b) Simulation parameters (α=1500 and UNIT=10,000).

Figure C.5: Network simulation: average time (in ms) needed to process a packet, using GCM(64k) Schemes.

Appendix D

Disk Encryption Performance Results

The complete reported results of the benchmarking application are listed in the following tables. The columns **Min**, **Max**, **Avg** and **SD**, denote the minimum, the maximum, the average and the standard deviation of the measurement points, respectively.

Table D.1: Results of Disk Performance Simulation (Test=Encrypt, Key=128, Step=512 byte, Loops=1000).

	Min	Max	Avg	SD
ECB	28.58	287.69	246.12	50.37
CBC	19.32	266.48	227.69	46.06
SCC	19.25	264.37	226.62	47
ESCC	19.26	233.9	200.65	40.52
LRW	16.92	135.12	121.57	18.82
XTS	19.11	187.4	164.87	30.03
MCB	19.11	250.9	215	44.2
ELEPHANT	18.63	174.5	154.13	27.07
ELEPHANT+	18.31	171.74	152.67	26.08
ELEPHANT*	18.69	184.72	162.18	31.72

Table D.2: Results of Disk Performance Simulation (Test=Decrypt, Key=128, Step=512 byte, Loops=1000).

	Min	Max	Avg	SD
ECB	19.19	284.51	241.9	50.33
CBC	19.07	263.39	224.03	46.89
SCC	19.23	262.51	225.46	45.55
ESCC	19.17	231.65	199.91	39.88
LRW	16.88	136.41	122.46	18.89
XTS	19.03	185.67	163.43	29.91
MCB	18.23	249.56	215.36	42.89
ELEPHANT	18.38	172.62	151.67	29.18
ELEPHANT+	18.26	170.26	150.04	26.77
ELEPHANT*	18.75	183.69	160.38	29.55

Table D.3: Results of Disk Performance Simulation (Test=Encrypt+Write, Key=128, Step=512 byte, Loops=1000).

	Min	Max	Avg	SD
ECB	16.05	257.02	214.15	45.49
CBC	15.93	239.09	202.96	41.01
SCC	14.72	238.01	198.31	44.99
ESCC	15.96	212.34	181.22	35.6
LRW	14.2	128.31	113.79	18.73
XTS	15.82	173.54	150.76	27.9
MCB	16.06	229.41	193.33	39.7
ELEPHANT	15.2	161.4	141.26	25.35
ELEPHANT+	14.97	159.51	138.84	25.47
ELEPHANT*	15.23	170.84	147.52	27.68

Table D.4: Results of Disk Performance Simulation (Test=Read+Decrypt, Key=128, Step=512 byte, Loops=1000).

	Min	Max	Avg	SD
ECB	16.06	252.9	212.58	43.51
CBC	15.88	235.61	196.27	42.5
SCC	15.9	234.62	197.34	40.48
ESCC	18.58	209.84	178.02	34.3
LRW	14.15	127.77	112.77	19.84
XTS	15.48	171.81	147.92	27.22
MCB	15.78	225.36	190.83	39.08
ELEPHANT	14.27	158.33	136.94	25.91
ELEPHANT+	14.8	157.74	135.88	26.29
ELEPHANT*	14.52	167.63	143.78	28.51

Table D.5: Results of Disk Performance Simulation (Test=Encrypt, Key=256, Step=512 byte, Loops=1000).

	Min	Max	Avg	SD
ECB	19.25	212.79	184.86	35.51
CBC	19.16	199.69	174.48	32.01
SCC	19.12	197.18	172.56	31.29
ESCC	19.06	179.91	158.85	27.64
LRW	15.7	116.29	105.36	15.4
XTS	17.8	151.48	135.2	22.01
MCB	19.11	189.83	167.75	28.44
ELEPHANT	17.08	141.12	126.49	20.21
ELEPHANT+	16.77	139.8	124.95	20.73
ELEPHANT*	17.33	147.9	132.42	21.03

Table D.6: Results of Disk Performance Simulation (Test=Decrypt, Key=256, Step=512 byte, Loops=1000).

	Min	Max	Avg	SD
ECB	19.24	211.78	185.13	33.44
CBC	19.11	197.98	173.24	31.46
SCC	19.06	196.94	172.98	31.95
ESCC	18.94	178.61	157.09	27.95
LRW	15.73	116.39	105.25	15.59
XTS	17.68	150.65	134.53	22.28
MCB	19.17	189	166.67	30.33
ELEPHANT	16.79	140.64	124.86	22.5
ELEPHANT+	16.7	138.88	124.34	20.32
ELEPHANT*	17.08	147.44	131.42	21.97

Table D.7: Results of Disk Performance Simulation (Test=Encrypt+Write, Key=256, Step=512 byte, Loops=1000).

	Min	Max	Avg	SD
ECB	15.98	194.72	167.83	31.82
CBC	16.14	183.5	159.12	29.1
SCC	12.56	182.83	156.48	32.3
ESCC	15.38	167.58	144.85	27.46
LRW	13.26	111.04	98.74	15.84
XTS	14.81	142.32	125.34	21.01
MCB	15.75	177.17	153.25	28.41
ELEPHANT	14.08	132.61	117.55	19.87
ELEPHANT+	13.84	131.43	116.33	19.88
ELEPHANT*	14.09	139.2	122.38	21.35

Table D.8: Results of Disk Performance Simulation (Test=Read+Decrypt, Key=256, Step=512 byte, Loops=1000).

	Min	Max	Avg	SD
ECB	16.02	193.82	165.07	32.04
CBC	15.62	181.43	155.54	29.51
SCC	15.65	180.31	154.88	29.49
ESCC	15.36	165.24	142.74	26.79
LRW	13.23	110.31	98.64	15.7
XTS	14.27	141.14	123.03	21.6
MCB	15.52	175	151.08	28.93
ELEPHANT	13.74	130.75	114.65	19.96
ELEPHANT+	11.58	129.46	114	21.21
ELEPHANT*	13.95	137.39	120.38	20.7

Table D.9: Results of Disk Performance Simulation (Test=Encrypt, Key=128, Step=1 MB, Loops=10).

	Min	Max	Avg	SD
ECB	277.14	310.43	305.78	6.07
CBC	231.78	285.95	281.21	8.99
SCC	127.69	283.53	278.14	14.84
ESCC	212.5	247.52	243.92	5.79
LRW	126.87	140.56	138.74	2.29
XTS	176.5	196.65	194.44	3.25
MCB	233.52	267.88	263.95	5.82
ELEPHANT	161.3	182.28	179.89	3.14
ELEPHANT+	114.53	179.94	176.72	6.55
ELEPHANT*	156.92	194.18	191.22	4.52

Table D.10: Results of Disk Performance Simulation (Test=Decrypt, Key=128, Step=1 MB, Loops=10).

	Min	Max	Avg	SD
ECB	263.17	307.02	302.72	7.33
CBC	245.62	281.46	277.55	6.65
SCC	249.55	281.08	276.94	5.82
ESCC	219.12	246.32	242.89	4.92
LRW	129.86	140.56	139.15	2.13
XTS	172.84	195.27	192.62	4.25
MCB	236.59	266.48	262.89	5.26
ELEPHANT	160.45	181.08	178.64	2.92
ELEPHANT+	64.4	178.19	175.34	8.81
ELEPHANT*	171.09	192.87	190.12	3.55

Table D.11: Results of Disk Performance Simulation (Test=Encrypt+Write, Key=128, Step=1 MB, Loops=10).

	Min	Max	Avg	SD
ECB	39.99	85.36	61.05	4.21
CBC	56.84	75.2	60.14	3.42
SCC	56.42	75.13	60.04	3.35
ESCC	54.85	71.19	58.1	2.87
LRW	46.57	60.17	49	1.9
XTS	52.5	67.62	54.68	2.51
MCB	56.19	75.68	59.17	3.09
ELEPHANT	50.29	66.98	53.42	2.39
ELEPHANT+	50.51	67.6	53.22	2.46
ELEPHANT*	50.2	67.85	54.42	2.56

Table D.12: Results of Disk Performance Simulation (Test=Read+Decrypt, Key=128, Step=1 MB, Loops=10).

	Min	Max	Avg	SD
ECB	215.01	252.26	235.68	4.99
CBC	200.3	249.77	220.81	5.26
SCC	201.57	243.16	220.48	5.02
ESCC	181.31	216.36	197.77	4.27
LRW	112.86	132.3	123.57	2.29
XTS	151.33	175.9	164.56	3.14
MCB	192.5	239.05	213.02	5.48
ELEPHANT	132.58	162.39	153.8	2.97
ELEPHANT+	137.64	164.28	152.04	2.95
ELEPHANT*	149.73	175.91	162.26	3.04

Table D.13: Results of Disk Performance Simulation (Test=Encrypt, Key=256, Step=1 MB, Loops=10).

	Min	Max	Avg	SD
ECB	202.83	224.86	221.96	4.03
CBC	181.34	209.63	206.96	4.19
SCC	188.65	208.25	204.6	3.57
ESCC	167.8	188.75	186.2	3.38
LRW	109.19	119.86	118.74	1.92
XTS	142.23	157.74	155.68	3.07
MCB	181.87	200.27	197.72	3.33
ELEPHANT	129.82	146.41	144.55	2.78
ELEPHANT+	133.3	145.29	143.23	2.46
ELEPHANT*	138.52	154.3	152.2	2.66

Table D.14: Results of Disk Performance Simulation (Test=Decrypt, Key=256, Step=1 MB, Loops=10).

	Min	Max	Avg	SD
ECB	199.01	223.68	221.08	4.26
CBC	188.89	209.11	206.72	3.79
SCC	180.44	207.54	204.26	4.84
ESCC	167.98	187.41	184.83	3.53
LRW	110.06	119.94	118.85	1.74
XTS	141.62	156.8	154.95	2.81
MCB	180.68	199.32	196.95	3.64
ELEPHANT	131.27	146.01	144.26	2.41
ELEPHANT+	128.79	144.33	142.38	2.44
ELEPHANT*	135.09	153.59	151.38	2.86

Table D.15: Results of Disk Performance Simulation (Test=Encrypt+Write, Key=256, Step=1 MB, Loops=10).

	Min	Max	Avg	SD
ECB	52.88	73.68	56.75	3
CBC	40.19	68.14	55.53	2.83
SCC	53	67.96	55.44	2.3
ESCC	50.87	67.53	54.03	2.47
LRW	44.66	56.22	46.21	1.6
XTS	48.06	62.49	50.96	2.06
MCB	52.07	67.52	54.92	2.55
ELEPHANT	47.63	60.51	49.69	1.94
ELEPHANT+	47.53	60.55	49.54	1.9
ELEPHANT*	48.45	62.99	50.66	2.07

Table D.16: Results of Disk Performance Simulation (Test=Read+Decrypt, Key=256, Step=1 MB, Loops=10).

	Min	Max	Avg	SD
ECB	167.43	193.09	182.51	3.56
CBC	160.29	191	172.97	3.29
SCC	157.5	189.05	171.46	3.25
ESCC	146.9	171.6	158.05	3.03
LRW	99.21	112.28	106.81	1.8
XTS	123.4	146.02	135.06	2.73
MCB	151.2	184.15	167.15	3.64
ELEPHANT	117.96	135.22	127.59	2.16
ELEPHANT+	115.01	134.65	126.41	2.2
ELEPHANT*	124.67	142.7	133.51	2.26

Bibliography

[1] The IEEE Security in Storage Work Group. http://siswg.org/, 2008.

[2] FIPS PUB 186. Digital Signature Standard (DSS), 1994.

[3] C. Adams. Constructing Symmetric Ciphers Using the CAST Design Procedure. *Design, Codes and Cryptography*, 12(3), 1997.

[4] C. Adams and J. Gilchrist. The CAST-256 Encryption Algorithm. RFC 2612, 1999.

[5] R. Anderson and E. Biham. Two Practical and Provable Secure Block Ciphers: BEAR and LION. In *Proceedings of Fast Software Encryption*, 1996.

[6] R. Anderson, E. Biham, and L. Knudsen. Serpent: A Proposal for the Advanced Encryption Standard. In *Proceeding of the AES conference*, 1998.

[7] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A Concrete Security Treatment of Symmetric Encryption. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, 1997.

[8] M. Bellare, T. Krovetz, and P. Rogaway. Luby-Rackoff Backwards: Increasing Security By Making Block Ciphers Non-Invertible. In *Proceedings of Advances in Cryptology – EUROCRYPT*, 1998.

[9] M. Bellare, P. Rogaway, and D. Wagner. The EAX Mode of Operation. In *Proceedings of Fast Software Encryption*, 2004.

[10] L. Berger. RSVP over ATM Implementation Requirements. RFC 2380, 1998.

[11] D. Bernstein. Floating-Point Arithmetic and Message Authentication. http://cr.yp.to/papers.html, 1999.

[12] E. Biham. New Types of Cryptanalytic Attacks Using Related Keys. In *Proceedings of Advances in Cryptology – EUROCYRPT*, 1994.

[13] E. Biham. On Matsui's Linear Cryptanalysis. In *Proceedings of Advances in Cryptology – EUROCYRPT*, 1995.

[14] E. Biham, A. Biryukov, N. Ferguson, L. Knudsen, B. Schneier, and A. Shamir. Cryptanalysis of MAGENTA. `www.ii.uib.no/~larsr/papers/magenta.pdf`, 1999.

[15] E. Biham, A. Biryukov, and A. Shamir. Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. *Journal of Cryptology*, 18(4), 2005.

[16] E. Biham, O. Dunkelman, and N. Keller. The Rectangle Attack - Rectangling the Serpent. In *Proceedings of Advances in Cryptology – EUROCYRPT*, 2001.

[17] E. Biham, O. Dunkelman, and N. Keller. Related-key Boomerang and Rectangle Attacks. In *Proceedings of Advances in Cryptology – EUROCYRPT*, 2005.

[18] E. Biham, O. Dunkelman, and N. Keller. Related-Key Impossible Differential Attacks on AES-192. In *Proceedings of RSA Conference*, 2006.

[19] E. Biham and N. Keller. Cryptanalysis of Reduced Variants of Rijndael. In *Proceedings of the AES Conference*, 2000.

[20] E. Biham and A. Shamir. Differential Cryptanalysis of DES-Like Cryptosystems. In *Proceedings of Advances in Cryptology – CRYPTO*, 1991.

[21] A. Biryukov. Boomerang attack on 5- and 6-round AES. In *Proceedings of the AES Conference*, 2005.

[22] A. Biryukov and D. Wagner. Advanced Slide Attacks. In *Proceedings of Advances in Cryptology – EUROCYRPT*, 2000.

[23] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway. UMAC: Fast and Secure Message Authentication. In *Proceedings of Advances in Cryptology – CRYPTO*, 1999.

[24] M. Boesgaard, M. Vesterager, T. Pedersen, J. Christiansen, and O. Scavenius. Rabbit: A New High-Performance Stream Cipher. In *Proceedings of Fast Software Encryption*, 2003.

[25] J. Bonneau and I. Mironov. Cache-Collision Timing Attacks Against AES. In *Proceedings of Cryptographic Hardware and Embedded Systems*, 2006.

[26] D. Borman, S. Deering, and R. Hinden. IPv6 Jumbograms. RFC 2675, 1999.

[27] L. Brown and J. Pieprzyk. Introducing the New LOKI97 Block Cipher. In *Proceeding of the AES Candidate Conference*, 1998.

[28] C. Burwick, D. Coppersmith, E. D'Avignon, R. Gennaro, S. Halevi, C. Jutla, S. Matyas, L. O'Connor, M. Peyravian, D. Safford, and N. Zunic. MARS — A Candidate Cipher for AES. In *Proceeding of the AES conference*, 1998.

[29] G. Carter, E. Dawson, and L. Nielsen. Key Schedules of Iterative Block Ciphers. In *Proceedings of the Australasian Conference on Information Security and Privacy*, 1998.

[30] G. Carter, E. Dawson, and L. Nielsen. Key Schedule Classification of the AES Candidates. In *Proceedings of the AES Conference*, 1999.

[31] J. Cheon, M. Kim, K. Kim, J. Lee, and S. Kang. Improved Impossible Differential Cryptanalysis of Rijndael and Crypton. In *Proceedings of the International Conference Seoul on Information Security and Cryptology*, 2002.

[32] K. Chun, S. Kim, S. Lee, S. Hak Sung, and S. Yoon. Differential and Linear Cryptanalysis for 2-Round SPNs. *Information Processing Letters*, 87(5), 2003.

[33] D. Coppersmith, D. Johnson, and S. Matyas. A Proposed Mode for Triple-DES Encryption. *IBM Journal of Research and Development*, 40(2), 1996.

[34] J. Daemen. Limitations of the Even-Mansour Construction. In *Proceedings of Advances in Cryptology – ASIACRYPT*, 1991.

[35] J. Daemen and C. Clapp. Fast Hashing and Stream Encryption with PANAMA. In *Proceedings of Fast Software Encryption*, 1998.

[36] J. Daemen, L. Knudsen, and V. Rijmen. The Block Cipher Square. In *Proceedings of Fast Software Encryption*, 1997.

[37] J. Daemen and V. Rijmen. AES Proposal: Rijndael. http://citeseer.ist.psu.edu/daemen98aes.html, 1998.

[38] J. Daemen and V. Rijmen. *The Design of Rijndael.* Springer-Verlag New York, Inc., 2002.

[39] J. Davidson and S. Jinturkar. An Aggressive Approach to Loop Unrolling. Technical report, Department of Computer Science. University of Virginia. Charlottesville, 1995.

[40] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6), 1976.

[41] J. Dunn and C. Martin. Terminology for ATM Benchmarking. RFC 2761, 2000.

[42] M. El-Fotouh and K. Diepold. AES Cryptographic Modes of Operation for High-Speed Networks. In *Proceedings of the International Conference on High Performance Computing, Networking and Communication Systems*, 2007.

[43] M. El-Fotouh and K. Diepold. Distributed Computing: Windows and Linux. Dr.Dobb's Journal, 2007.

[44] M. El-Fotouh and K. Diepold. Statistical Testing for Disk Encryption Modes of Operations. Cryptology ePrint Archive, Report 2007/362, 2007.

[45] M. El-Fotouh and K. Diepold. A Fast Encryption Scheme for Networks Applications. In *Proceedings of SECRYPT*, 2008.

[46] M. El-Fotouh and K. Diepold. A New Narrow Block Mode of Operations for Disk Encryption. In *Proceedings of the International Conference on Information Assurance and Security*, 2008.

[47] M. El-Fotouh and K. Diepold. Dynamic Substitution Model. In *Proceedings of the International Conference on Information Assurance and Security*, 2008.

[48] M. El-Fotouh and K. Diepold. Galois Substitution Counter Mode (GSCM). In *Proceedings of the International Workshop on Security and Privacy in Enterprise Computing in conjunction with the IEEE International EDOC Conference*, 2008.

[49] M. El-Fotouh and K. Diepold. The Substitution Cipher Chaining Mode. In *Proceedings of SECRYPT*, 2008.

[50] M. El-Fotouh and K. Diepold. Breaking the Substitution Cipher Chaining Mode (SCC-256). In *Proceedings of the International Conference on Cryptography, Coding and Information Security*, 2009.

[51] M. El-Fotouh and K. Diepold. Cryptanalysis of Substitution Cipher Chaining Mode (SCC). In *Proceedings of IEEE International Conference on Communications*, 2009.

[52] M. El-Fotouh and K. Diepold. Dynamic Injection Model. In *Proceedings of International Conference on Cryptography, Coding and Information Security*, 2009.

[53] M. El-Fotouh and K. Diepold. Dynamic Permutation Model. In *Proceedings of International Conference on Information Security and Privacy*, 2009.

[54] M. El-Fotouh and K. Diepold. Extended Substitution Cipher Chaining Mode (ESCC). Cryptology ePrint Archive, Report 2009/182, 2009.

[55] M. El-Fotouh and K. Diepold. The Analysis of Galois Substitution Counter Mode (GSCM). Cryptology ePrint Archive, Report 2009/140, 2009.

[56] M. El-Fotouh and K. Diepold. The Pushdown Attack on AES. In *Proceedings of SECUREWARE*, 2009.

[57] M. El-Fotouh and K. Diepold. The Security of Dynamic and Static Substitution Models. In *Proceedings of International Conference on Information Security and Privacy*, 2009.

[58] S. Even and Y. Mansour. A Construction of a Cipher from a Single Pseudorandom Permutation. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 10(3), 1997.

[59] Federal Information Processing Standards Publication 46. Data Encryption Standard, 1977.

[60] N. Ferguson. AES-CBC + Elephant diffuser : A Disk Encryption Algorithm for Windows Vista. `http://download.microsoft.com/download/0/2/3/0238acaf-d3bf-4a6d-b3d6-0a0be4bbb36e/BitLockerCipher200608.pdf`, 2006.

[61] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting. Improved Cryptanalysis of Rijndael. In *Proceedings of Fast Software Encryption*, 2001.

[62] C. Fruhwirth. New Methods in Hard Disk Encryption. `http://clemens.endorphin.org/nmihde/nmihde-A4-ds.pdf`, 2005.

[63] K. Gaj and P. Chodowiec. Hardware Performance of the AES Finalists - Survey and Analysis of Results. http://ece.gmu.edu/crypto/AES_survey.pdf, 2000.

[64] T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Proceedings of Advances in Cryptology – CRYPTO*, 1985.

[65] H. Gilbert and M. Minier. A Collision Attack on 7 Rounds of Rijndael. In *Proceedings of the AES Conference*, 2000.

[66] B. Gladman. `http://fp.gladman.plus.com/AES/index.htm`, 2008.

[67] L. Granboulan, P. Nguyen, F. Noilhan, and S. Vaudenay. DFCv2. In *Proceedings of Selected Areas in Cryptography*, 2001.

[68] C. Greg. The Nature of the Beast: Recent Traffic Measurements from an Internet Backbone. http://citeseer.ist.psu.edu/673025.html, 1998.

[69] D. Grossman and J. Heinanen. Multiprotocol Encapsulation over ATM Adaptation Layer 5. RFC 2684, 1999.

[70] S. Gueron. Advanced Encryption Standard (AES) Instructions Set. http://softwarecommunity.intel.com/articles/eng/3788.htm, 2008.

[71] S. Halevi. EME*: Extending EME to Handle Arbitrary-Length Messages With Associated Data. http://citeseer.ist.psu.edu/halevi04eme.html, 2004.

[72] S. Halevi. Invertible Universal Hashing and the TET Encryption Mode. Cryptology ePrint Archive, Report 2007/014, 2007.

[73] S. Halevi and H. Krawczyk. Security Under Key-Dependent Inputs. In *Proceedings of the ACM conference on Computer and communications security*, 2007.

[74] S. Halevi and P. Rogaway. A Parallelizable Enciphering Mode. Cryptology ePrint Archive, Report 2003/147.

[75] S. Halevi and P. Rogaway. A Tweakable Enciphering Mode. Cryptology ePrint Archive, Report 2003/148.

[76] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409. Technical report, 1998.

[77] J. Hastad. The Security of the IAPM and IACBC Modes. *Journal of Cryptology*, 20(2), 2007.

[78] S. Hong, J. Kim, S. Lee, and B. Preneel. Related-Key Rectangle Attacks on Reduced Versions of SHACAL-1 and AES-192. In *Proceedings of Fast Software Encryption*, 2005.

[79] S. Hong, S. Lee, J. Lim, J. Sung, D. Cheon, and I. Cho. Provable Security Against Differential and Linear Cryptanalysis for the SPN Structure. In *Proceedings of Fast Software Encryption*, 2001.

[80] R. Housley. Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP). RFC 3686, 2004.

[81] K. Huber. The MAGENTA Block Cipher Algorithm. In *Proceedings of the AES Conference*, 1998.

[82] IEEE P1619. Draft Proposal for Tweakable Narrow-block Encryption. `http://www.siswg.org/docs/LRW-AES-10-19-2004.pdf`, 2004.

[83] G. Jakimoski and Y. Desmedt. Related-Key Differential Cryptanalysis of 192-bit Key AES Variants. In *Proceedings of Selected Areas in Cryptography*, 2004.

[84] O. Jung, S. Kuhn, C. Ruland, and K. Wollenweber. Enhanced Modes of Operation for the Encryption in High-Speed Networks and Their Impact on QoS. In *Proceedings of the Australasian Conference on Information Security and Privacy*, 2001.

[85] L. Keliher. Refined Analysis of Bounds Related to Linear and Differential Cryptanalysis for the AES. In *Proceedings of the Advanced Encryption Standard*, 2005.

[86] L. Keliher, H. Meijer, and S. Tavares. Improving the Upper Bound on the Maximum Average Linear Hull Probability for Rijndael. In *Proceedings of Selected Areas in Cryptography*, 2001.

[87] L. Keliher, H. Meijer, and S. Tavares. New Method for Upper Bounding the Maximum Average Linear Hull Probability for SPNs. In *Proceedings of Advances in Cryptology – EUROCYRPT*, 2001.

[88] L. Keliher and J. Sui. Exact Maximum Expected Differential and Linear Probability for Two-Round Advanced Encryption Standard. *Information Security, IET*, 1(2), 2007.

[89] J. Kelsey and B. Schneier. Key-Schedule Cryptanalysis of DEAL. In *Proceedings of Selected Areas in Cryptography*, 2000.

[90] J. Kelsey, B. Schneier, and D. Wagner. Related-key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA. In *Proceedings of the International Conference on Information and Communication Security*, 1997.

[91] S. Kent and R. Atkinson. IP Authentication Header. RFC 2402, 1998.

[92] S. Kent and R. Atkinson. IP Encapsulating Security Payload (ESP). RFC 2406, 1998.

[93] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, 1998.

[94] J. Kim, S. Hong, and B. Preneel. Related-Key Rectangle Attacks on Reduced AES-192 and AES-256. In *Proceedings of Fast Software Encryption*, 2007.

[95] J. Kim, G. Kim, S. Hong, S. Lee, and D. Hong. The Related-Key Rectangle Attack - Application to SHACAL-1. In *Proceedings of Information Security and Privacy*, 2004.

[96] L. Knudsen and V. Rijmen. On the Decorrelated Fast Cipher (DFC) and Its Theory. In *Proceedings of Fast Software Encryption*, 1999.

[97] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177), 1987.

[98] L. Hars. Public Comments on the XTS-AES Mode. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/XTS_comments-Hars.pdf`, 2008.

[99] RSA Laboratories. The RC6 Block Cipher. In *Proceeding of the AES conference*, 1998.

[100] X. Lai and J. Massey. A Proposal for a New Block Encryption Standard. In *Proceedings of Advances in Cryptology – EUROCYRPT*, 1991.

[101] Y. Lai, L. Chang, L. Chen, C. Chou, and C. Chiu. A Novel Memoryless AES Cipher Architecture for Networking Applications. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2004.

[102] H. Li and J. Li. A High Performance Sub-Pipelined Architecture for AES. In *Proceedings of the International Conference on Computer Design*, 2005.

[103] T. Liang, Y. Liu, and C. Shieh. Adding Memory Resource Consideration into Workload Distribution for Software DSM Systems. In *Proceeding of CLUSTER*, 2003.

[104] C. Lim. CRYPTON: A New 128-bit Block Cipher. In *Proceeding of the AES conference*, 1998.

[105] M. Liskov, R. Rivest, and D. Wagner. Tweakable Block Ciphers. In *Proceedings of Advances in Cryptology – CRYPTO*, 2002.

[106] S. Lucks. On Security of the 128-Bit Block Cipher DEAL. In *Proceedings of Fast Software Encryption*, 1999.

[107] S. Lucks. Attacking Seven Rounds of Rijndael under 192-bit and 256-bit Keys. In *Proceeding of the AES conference*, 2000.

[108] M. Ball. NIST's Consideration of XTS-AES as standardized by IEEE Std 1619-2007. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/collected_XTS_comments.pdf`, 2008.

[109] J. Massey, G. Khachatrian, and M. Kuregian. Nomination of SAFER+ as Candidate Algorithm for the Advanced Encryption Standard (AES). In *Proceeding of the AES conference*, 1998.

[110] M. Matsui. Linear Cryptanalysis Method for DES Cipher. In *Proceedings of Advances in Cryptology – EUROCYRPT*, 1994.

[111] M. Matsui and T. Tokita. Cryptanalysis of a Reduced Version of the Block Cipher E2. In *Proceedings of Fast Software Encryption*, 1999.

[112] L. May, M. Henricksen, W. Millan, G. Carter, and E. Dawson. Strengthening the Key Schedule of the AES. In *Proceedings of the Australian Conference on Information Security and Privacy*, 2002.

[113] D. McGrew. Counter Mode Security: Analysis and Recommendations. http://citeseer.ist.psu.edu/mcgrew02counter.html, 2002.

[114] D. McGrew and S. Fluhrer. The Extended Codebook (XCB) Mode of Operation. Cryptology ePrint Archive, Report 2004/278, 2004.

[115] D. McGrew and J. Viega. Arbitrary block length mode. http://grouper.ieee.org/groups/1619/email/pdf00005.pdf, 2004.

[116] D. McGrew and J. Viega. The Galois/Counter Mode of Operation (GCM). http://citeseer.ist.psu.edu/mcgrew04galoiscounter.html, 2004.

[117] A. Menezes, P. Van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[118] V. Miller. Use of Elliptic Curves in Cryptography. In *Proceedings of Advances in Cryptology – CRYPTO*, 1986.

[119] S. Mister and S. Tavares. Cryptanalysis of RC4-like Ciphers. In *Proceedings of Selected Areas in Cryptography*, 1999.

[120] National Institute of Standards and Technology. Advanced Encryption Standard, NIST FIPS PUB 197, 2001.

[121] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, and E. Roback. Report on the Development of the Advanced Encryption Standard (AES). Technical report, 2000.

[122] M. Neve and J. Seifert. Advances on Access-Driven Cache Attacks on AES. In *Proceedings of Selected Areas in Cryptography*, 2006.

[123] M. Neve, J. Seifert, and Z. Wang. A Refined Look at Bernstein's AES Side-Channel Analysis. In *Proceedings of the ACM Symposium on Information, computer and communications security*, 2006.

[124] NIST. Status Report on the First Round of the Development of the Advanced Encryption Standard. `http://nvl.nist.gov/pub/nistpubs/jres/104/5/j45nec.pdf`, 1999.

[125] NIST. Public Comments on AES Candidate Algorithms - Round 2. `http://csrc.nist.gov/archive/aes/round2/comments/R2comments.txt`, 2000.

[126] NIST. Advanced Encryption Standard (AES) Development Effort. `http://csrc.nist.gov/archive/aes/index.html`, 2001.

[127] NIST. Announcing the ADVANCED ENCRYPTION STANDARD (AES). Technical Report 197, Federal Information Processing Standards Publication, 2001.

[128] NIST. Guide to Storage Encryption Technologies for End User Devices. `http://csrc.nist.gov/publications/nistpubs/800-111/SP800-111.pdf`, 2007.

[129] NIST. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. NIST Special Publication 800-38D, 2007.

[130] NIST. Public Comments on the XTS-AES Mode. `http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/collected_XTS_comments.pdf`, 2008.

[131] NIST. National Institute of Standards and Technology. `http://www.nist.gov/`, 2009.

[132] K. Nyberg and L. Knudsen. Provable Security Against a Differential Attack. *Journal of Cryptology*, 8(1), 1995.

[133] E. Oswald, J. Daemen, and V. Rijmen. AES - The State of the Art of Rijndael's Security. `http://www.iaik.tugraz.at/aboutus/people/oswald/papers/aes_report.pdf`, 2002.

[134] IEEE P1619. IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices. *IEEE Std 1619-2007*, 2008.

[135] S. Park, S. Sung, S. Chee, E. Yoon, and J. Lim. On the Security of Rijndael-Like Structures Against Differential and Linear Cryptanalysis. In *Proceedings of Advances in Cryptology – ASIACRYPT*, 2002.

[136] R. Phan. Impossible Differential Cryptanalysis of 7-Round Advanced Encryption Standard (AES). *Information Processing Letters*, 91(1), 2004.

[137] G. Poupard and S. Vaudenay. Decorrelated Fast Cipher: An AES Candidate Well Suited for Low Cost Smart Card Applications. In *Proceedings of the International Conference on Smart Card Research and Applications*, 2000.

[138] R. Rivest. The RC5 Encryption Algorithm. `http://people.csail.mit.edu/rivest/Rivest-rc5.pdf`, 1995.

[139] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2), 1978.

[140] P. Rogaway. The Security of DESX. http://citeseer.ist.psu.edu/rogaway96security.html, 1996.

[141] P. Rogaway. Efficient Instantiations of Tweakable Block ciphers and Refinements to Modes OCB and PMAC. http://citeseer.ist.psu.edu/ rogaway03efficient.html, 2003.

[142] P. Rogaway, M. Bellare, and J. Black. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. *ACM Transactions on Information and System Security*, 6(3), 2003.

[143] F. Sano, K. Ohkuma, H. Shimizu, and S. Kawamura. On the Security of Nested SPN Cipher against the Differential and Linear Cryptanalysis. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(1), 2003.

[144] B. Schneier. Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish). In *Proceedings of Fast Software Encryption*, 1994.

[145] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson. The Twofish Encryption Algorithm: A 128-bit Block Cipher. www.schneier.com/paper-twofish-paper.pdf, 1999.

[146] R. Schroeppel and H. Orman. Introduction to the Hasty Pudding Cipher. In *Proceeding of the AES conference*, 1998.

[147] C. Shannon. Communication Theory of Secrecy Systems. *Bell System Technical Journal*, 28(4), 1949.

[148] V. Shoup. On Fast and Provably Secure Message Authentication Based on Universal Hashing. In *Proceedings of Advances in Cryptology – CRYPTO*, 1996.

[149] R. Sinha, C. Papadopoulos, and J. Heidemann. Internet Packet Size Distributions: Some Observations. Technical Report ISI-TR-2007-643, USC/Information Sciences Institute, 2007.

[150] N. Sklavos, N. A. Moldovyan, and O. Koufopavlou. High Speed Networking Security: Design and Implementation of Two New DDP-Based Ciphers. *Mobile Networks Applications*, 10(2), 2005.

[151] J. Soto and L. Bassham. Randomness Testing of the Advanced Encryption Standard Finalist Candidates. Technical report, 2000.

[152] Internet World Stats. World Internet Usage and Population Statistics. http://www.internetworldstats.com/stats.htm, 2009.

[153] K. Tesink. Definitions of Managed Objects for ATM Management. RFC 2515, 1999.

[154] S. Tillich and J. Grossschaedl. Instruction Set Extensions for Efficient AES Implementation on 32-bit Processors. In *Proceedings of Cryptographic Hardware and Embedded Systems*, 2006.

[155] Trusted Computing Group. TCG TPM Specification Version 1.2. http://www.trustedcomputinggroup.org, 2009.

[156] D. Wagner. The Boomerang Attack. In *Proceedings of Fast Software Encryption*, 1999.

[157] D. Wagner, N. Ferguson, and B. Schneier. Cryptanalysis of Frog. `http://www.counterpane.com/frog.html`, 1998.

[158] D. Whiting, R. Housley, and N. Ferguson. Counter with CBC-MAC (CCM). RFC3610, 2003.

[159] W. Zhang, W. Wu, and D. Feng. New Results on Impossible Differential Cryptanalysis of Reduced AES. In *Proceedings of the Information Security and Cryptology*, 2007.

[160] W. Zhang, W. Wu, and L. Zhang. Improved Related-Key Impossible Differential Attacks on Reduced-Round AES-256. http://www.lois.cn/LOIS-AES/data/AES-256.pdf, 2007.

[161] W. Zhang, W. Wu, L. Zhang, and D. Feng. Improved Related-Key Impossible Differential Attacks on Reduced-Round AES-192 . In *Proceedings of Selected Areas in Cryptography*, 2007.

List of Figures

List of Tables