Disk Encryption using parallelization of Block Cipher

A dissertation submitted in the partial fulfillment for the award of Degree of
Master of Technology
In
Software Technology

By

Bappa Mondal
(Roll no. 2K13/SWT/06)

Under the guidance of

Manoj Kumar

DEPARTMENT OF SOFTWARE ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY

BAWANA ROAD, DELHI

DECLARATION

I hereby want to declare that the thesis entitled "**Disk Encryption using parallelization of Block Cipher"** which is being submitted to the **Delhi Technological University**, in partial fulfillment of the requirements for the award of degree in **Master of Technology in Software Technology** is an authentic work carried out by me. The material contained in this thesis has not been submitted to any institution or university for the award of any degree.

Bappa Mondal
Computer Science and Engineering Department,
Delhi Technological University, Delhi

CERTIFICATE

Delhi Technological University
(Government of Delhi NCR)
Bawana Road, New Delhi-42

This is to certify that the thesis entitled **"Disk Encryption using parallelization of Block Cipher"** done by **BAPPA MONDAL** (Roll Number: **2K13/SWT/06**) for the partial fulfillment of the requirements for the award of degree of **Master of Technology** Degree in **Software Technology** in the **Department of Computer Science and Engineering**, Delhi Technological University, New Delhi is an authentic work carried out by him under my guidance.

Project Guide:
Manoj Kumar
Associate Professor
Department of Computer Science and Engineering
Delhi Technological University, Delhi

ACKNOWLEDGEMENT

I take this opportunity to express my deep sense of gratitude and respect towards my guide **Manoj Kumar, Associate Professor, Department of Computer Engineering.**

I am very much indebted to him for his generosity, expertise and guidance which I received from him while working on this project. Without his support and timely guidance the completion of the project would have seemed a far –fetched dream. In this respect I find myself lucky to have my guide. He has guided not only with the subject matter, but also taught the proper style and techniques of documentation and presentation.

Besides my guide, I would like to thank entire teaching and non-teaching staff in the Department of Computer Engineering, DTU for all their help during my tenure at DTU.

BAPPA MONDAL
M.Tech Software Technology
2K13/SWT/06

ABSTRACT

These days digital contents like contacts, emails, messages, images, videos, memo etc. are so important that we don't want to compromise those data if we lost our phone or some unauthorized access to our phone. In this paper, we will learn about Disk Encryption, why it is important, proposal of a customized block cipher and how it would be efficient
Comparatively available block cipher.

Disk encryption is a process where all user data on a device using an encrypted key. In a encrypted device, All data in write operation, encrypted before committing it to disk and all reads automatically decrypted before returning it to the calling process, also known as on-the-fly encryption. It means it always encrypt data in disk, just decrypt block when required and encrypt for newly requested block.

'Data at Rest' refers to data stored in persistent storage like Hard disk, eMMC, NandFlash where 'Data in Use' generally refers to data being processed by a CPU or in memory like DRAM.
Disk Encryption is a process of DAR, where we protect data stored on disk.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Disk encryption uses an encryption algorithm to convert every bit of data that goes to disk to Cipher text, ensuring that data cannot be read from the disk without the decryption key. Full-disk encryption (FDE) promises that everything on disk is encrypted, including operating system files, cache, and temporary files. In practice, a small part of the OS, or a separate OS loader, must be kept unencrypted so that it can obtain the decryption key and then decrypt and mount the disk volume(s) used by the main OS. The disk decryption key is usually stored encrypted and requires an additional key encryption key (KEK) in order to be decrypted. The KEK can either be stored in a hardware module, such as a smart card or a TPM, or derived from a passphrase obtained from the user on each boot. When stored in a hardware module, the KEK can also be protected by a user-supplied PIN or password.

This report proposes the methods which should be applied for faster encryption and less power consumption.

1.1. GENERAL

Disk Encryption: is a technology which protects information by converting it into unreadable code that cannot be deciphered easily by unauthorized people.

Data at rest (DAR): is an information technology term referring to inactive data which is stored physically in any digital form (e.g. databases, data warehouses, spreadsheets, archives, tapes, off-site backups, mobile devices etc.).

Types of DAR:

- **Disk encryption**

  Full-disk encryption (FDE) is encryption at the hardware level. FDE works by automatically converting data on a hard drive into a form that cannot be understood by anyone who doesn't have the key to "undo" the conversion. Without the proper authentication key, even if the hard drive is removed and placed in another machine, the data remains inaccessible. FDE can be installed on a computing device at the time of manufacturing or it can be added later on by installing a special software driver.

  The advantage of FDE is that it requires no special attention on the part of the end user after he initially unlocks the computer. As data is written, it is automatically encrypted. When it is read, it is automatically decrypted. Because everything on the hard drive is encrypted, including the operating system, a disadvantage of FDE is that the encrypting/decrypting process can slow down data access times, particularly when virtual memory is being heavily accessed.

- **Filesystem level encryption :**

  Filesystem-level encryption, often called file/folder encryption, is a form of disk encryption where individual files or directories are encrypted by the file system itself.

  This is in contrast to full disk encryption where the entire partition or disk, in which the file system resides, is encrypted.

  Types of filesystem-level encryption include:
  the use of a 'stackable' cryptographic filesystem layered on top of the main file system
  a single general-purpose file system with encryption

  The advantages of filesystem-level encryption include:

flexible file-based key management, so that each file can be and usually is encrypted with a separate encryption key individual management of encrypted files e.g. incremental backups of the individual changed files even in encrypted form, rather than backup of the entire encrypted volume. access control can be enforced through the use of public-key cryptography, and the fact that cryptographic keys are only held in memory while the file that is decrypted by them is held open.

- **Cipher Mode:**

Secondary storage like eMMC is divided into blocks. A block is the smallest readable or writable unit which can be addressed, typically size of 4KB. Disk Encryption uses same key for encrypting each blocks with AES algorithm (Symmetric Algorithm). Assume two separate blocks having same plain text, as we use same disk encryption key for all blocks, cipher text on those blocks will be same. This expose to Malleability attack, where plaintext is known to the adversary, it is possible to change every 2nd plaintext block to a value chosen by the attacker, while the blocks in between are changed to random values, which does not ensure integrity of the encrypted data.

To avoid such compromise on data integrity, Disk Encryption uses block cipher based modes, such as CBC (Cipher-block chaining), where previous block's cipher text is x-or ed with the current block's plaintext before encryption. Therefore for same plain test in different block, cipher test will be different.

1.2. MOTIVATION

With CBC cipher block, we solved malleability attack, but again it creates a problem when encrypting/decrypting a particular block, because blocks are accessed non-sequentially, it need to encrypt / decrypt whole disk. Disk access cannot depend on the contents of their preceding/succeeding sectors.

To avoid such re-work of encryption, CBC algorithm requires initialization vector (IV). Each sector need to be independent of each other and somehow random in nature and unpredictable. Therefore each block requires a separate IV to support random block read/write. The usual methods for generating IVs are predictable sequences of numbers, for example, time stamp or sector number and permit certain attack such as a watermarking attack. Watermarking attack is an attack on disk encryption methods where the presence of a specially crafted piece of data can be detected by an attacker without knowing the encryption key. If these IVs are predictable by an attacker (and the file system reliably starts the content at the same offset to the start of each sector, and files are likely to be largely contiguous), then there is a chosen plaintext attack which can reveal the existence of encrypted data.

With the proposed paper, we will see how we can generate IV for each individual block and how we can achieve parallelization on it. With the advancement of Multi-processor, we can achieve significant amount of faster encryption as well as save power consumption.

1.3. PROBLEM STATEMENT

From the forgoing section we can see that for Disk Encryption, we generally use symmetric key encryption like AES and since its works on block level, we need to use some block cipher. One of the widely used block cipher is CBC (Cipher-block chaining), where previous block's cipher text is x-or ed with the current block's plaintext before encryption.

Problem with CBC is during encryption, whole disk need to encrypt at the same time, also parallelization is not possible here.

This leads to the approach of detailed study of block cipher used in disk encryption and how to achieve parallelization for faster speed and independent block encryption.

This thesis aims at studying various block cipher and provide some way to achieve parallelization.

To establish the performance, a statistical technique used to find relationships between the amount of data to encrypt and time spent on it during both CBC and proposed parallel block cipher.

More specifically, In CBC mode, decryption is independent of each block, therefore. Parallelization can be achieved; therefore, performance evaluation is performed during encryption only.

1.4. SCOPE OF THE THESIS

The scope of this thesis is to study the Disk Encryption, related block cipher and AES algorithm. Thesis proposes combination of block cipher and AES encryption in disk encryption. With exiting block cipher, we can’t find parallelization, here, we will see with modification of block cipher, how we can achieve parallelization in disk encryption.

During encryption, encryption block size taken as 512 bytes, where as standard Linux file system block size(EXT4) is 4KB. Therefore, we can expect better performance/faster performance in 4KB block size. This paper has some basic details of AES encryption algorithm, but details about Block cipher and its comparison, Proposal of new block cipher and its implementation and advantages.

Reason we tool encryption block size as 512 bytes, as reference with Android Disk Encryption.

1.5. THESIS ORGANIZATION

Chapter 1 Begins with General introduction and related work. It addresses the topics like Motivation, Problem Statement, Scope, Related work and thesis organization.

Chapter 2 Provides a detailed description about various encryption techniques and gives a details about present block cipher and brief of various attacks.

Chapter 3 Presents the proposed research methodology which explains the detailed model of modes of blocks cipher used in disk encryption. how parallelization can be achieve.

Chapter 4 presents the results and analysis part of the proposed methodology.

Chapter 5 concludes the thesis.

CHAPTER 2
LITERATURE REVIEW

2.1. Symmetric Encryption:

Symmetric encryption is the oldest and best-known technique. A secret key, which can be a number, a word, or just a string of random letters, is applied to the text of a message to change the content in a particular way. This might be as simple as shifting each letter by a number of places in the alphabet. As long as both sender and recipient know the secret key, they can encrypt and decrypt all messages that use this key.

Popular symmetric algorithms include Twofish, Serpent, AES (Rijndael), Blowfish, CAST5, Grasshopper, RC4, 3DES, Skipjack, Safer+/++ (Bluetooth), and IDEA.

2.2. Asymmetric Encryption:

The problem with secret keys is exchanging them over the Internet or a large network while preventing them from falling into the wrong hands. Anyone who knows the secret key can decrypt the message. One answer is asymmetric encryption, in which there are two related keys--a key pair. A public key is made freely available to anyone who might want to send you a message. A second, private key is kept secret, so that only you know it. Any message (text, binary files, or documents) that are encrypted by using the public key can only be decrypted by applying the same algorithm, but by using the matching private key. Any message that is encrypted by using the private key can only be decrypted by using the matching public key. Popular symmetric algorithms include RSA.

2.3. Advanced Encryption Standard (AES):

The Advanced Encryption Standard (AES), also known as Rijndael, is a specification for the encryption technique by the U.S. National Institute of Standards and Technology (NIST). AES is a subset of the Rijndael cipher developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen. For AES, each with a block size of 128 bits, but three different key lengths: 128, 192 and 256 bits. The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used for both encrypting and decrypting the data.

High-level description of the algorithm:

- KeyExpansions—round keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate 128-bit round key block for each round plus one more.
- InitialRound
  - AddRoundKey—each byte of the state is combined with a block of the round key using bitwise xor.
- Rounds
  - SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.
  - ShiftRows—a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
  - MixColumns—a mixing operation which operates on the columns of the state, combining the four bytes in each column.
  - AddRoundKey
- Final Round (no MixColumns)
  - SubBytes
  - ShiftRows
  - AddRoundKey.

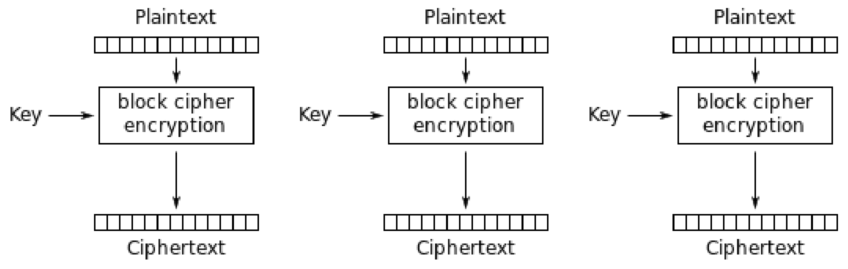
2.4. Block Cipher mode of operation:

In cryptography, a mode of operation is an algorithm that uses a block cipher to encrypt messages of arbitrary length in a way that provides confidentiality or authenticity. A block cipher by itself is only suitable for the secure cryptographic transformation (encryption or decryption) of one fixed-length group of bits called a block. A mode of operation describes how to repeatedly apply a cipher's single-block operation to securely transform amounts of data larger than a block.

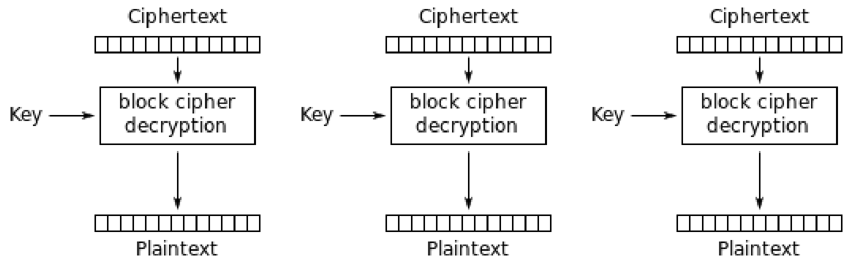
Common modes:

1. Electronic Codebook (ECB)

The simplest of the encryption modes is the Electronic Codebook (ECB) mode. The message is divided into blocks, and each block is encrypted separately.



Electronic Codebook (ECB) mode encryption



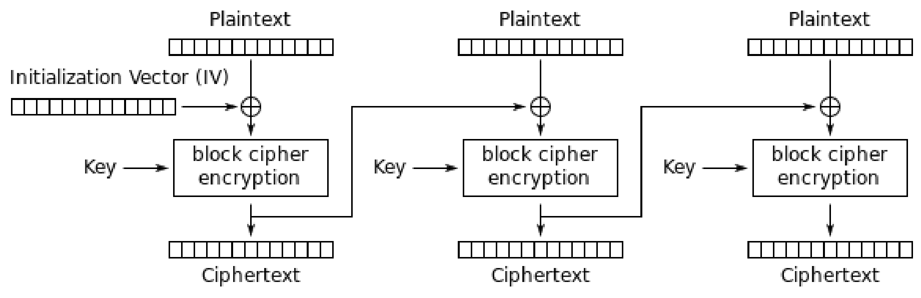
Electronic Codebook (ECB) mode decryption

Encryption parallelizable:	Yes
Decryption parallelizable:	Yes
Random read access:	Yes

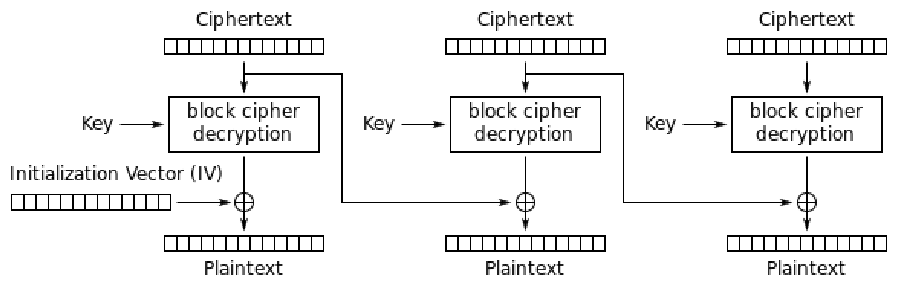
The disadvantage of this method is that identical plaintext blocks are encrypted into identical cipher text blocks.

2. Cipher Block Chaining (CBC)

Each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all plaintext blocks processed up to that point.



Cipher Block Chaining (CBC) mode encryption



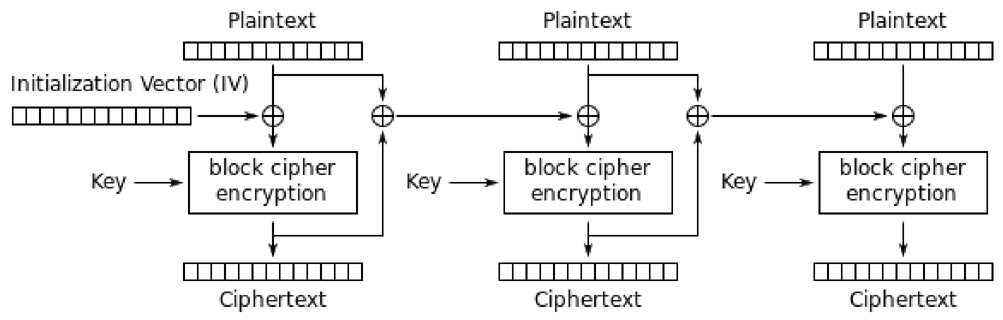
Cipher Block Chaining (CBC) mode decryption

Encryption parallelizable:	No
Decryption parallelizable:	Yes
Random read access:	Yes

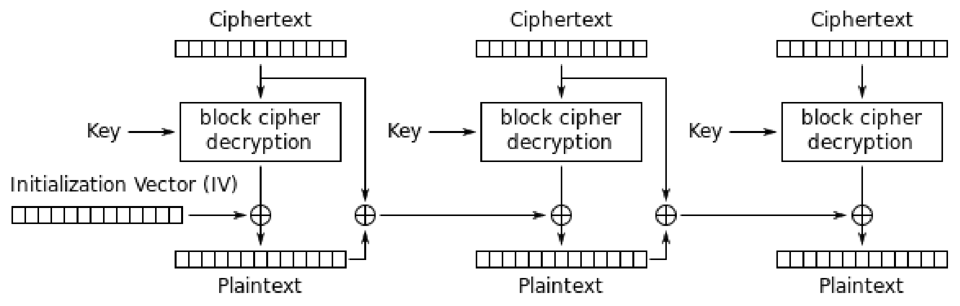
Its main drawbacks are that encryption is sequential (i.e., it cannot be parallelized), and that the message must be padded to a multiple of the cipher block size. One way to handle this last issue is through the method known as cipher text stealing. Note that a one-bit change in a plaintext or IV affects all following cipher text blocks.

3. Propagating Cipher Block Chaining (PCBC)

In PCBC mode, each block of plaintext is XORed with both the previous plaintext block and the previous cipher text block before being encrypted.



Propagating Cipher Block Chaining (PCBC) mode encryption



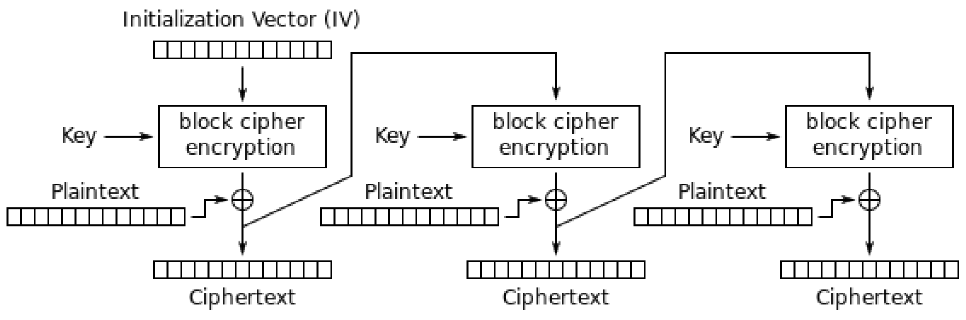
Propagating Cipher Block Chaining (PCBC) mode decryption

On a message encrypted in PCBC mode, if two adjacent cipher text blocks are exchanged, this does not affect the decryption of subsequent blocks.

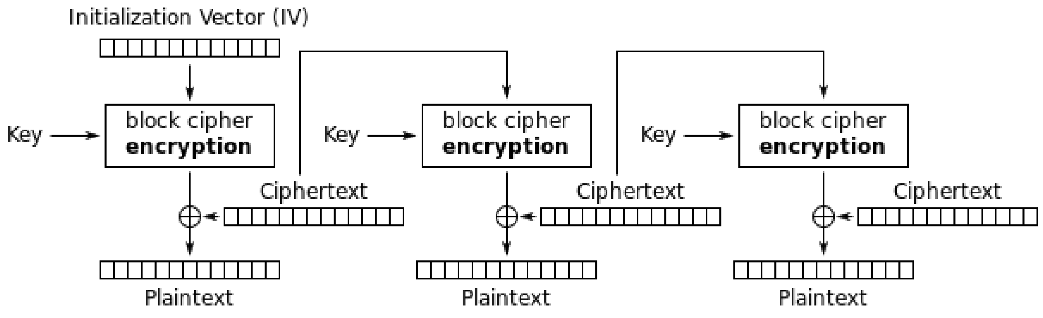
Encryption parallelizable:	No
Decryption parallelizable:	Yes
Random read access:	Yes

4. Cipher Feedback (CFB)

The Cipher Feedback (CFB) mode, a close relative of CBC, makes a block cipher into a self-synchronizing stream cipher. Operation is very similar; in particular,
CFB decryption is almost identical to CBC encryption performed in reverse.



Cipher Feedback (CFB) mode encryption



Cipher Feedback (CFB) mode decryption

Encryption parallelizable:	No
Decryption parallelizable:	Yes

Random read access: Yes

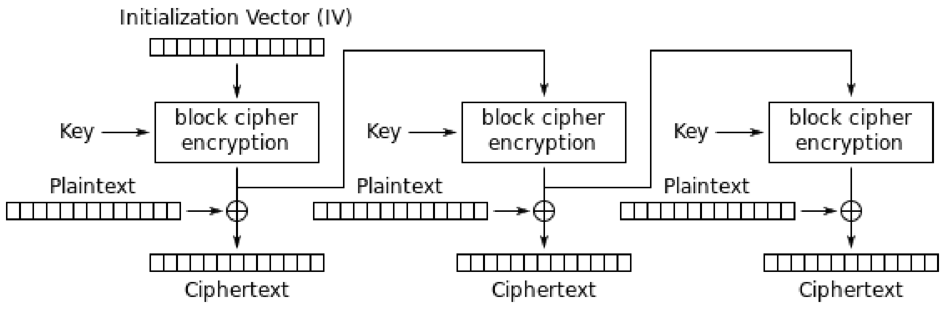
Like CBC mode, changes in the plaintext propagate forever in the cipher text, and encryption cannot be parallelized.

CFB shares two advantages over CBC mode with the stream cipher modes OFB and CTR:
the block cipher is only ever used in the encrypting direction, and the message

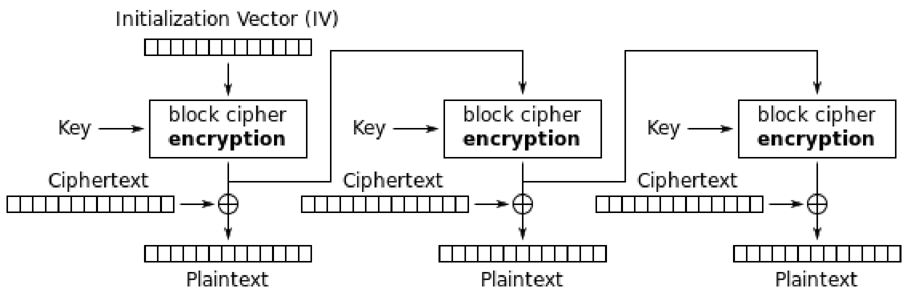
does not need to be padded to a multiple of the cipher block size.

5. Output Feedback (OFB)

It generates keystream blocks, which are then XORed with the plaintext blocks to get the cipher text. Just as with other stream ciphers, flipping a bit in the cipher text produces a flipped bit in the plaintext at the same location.



Output Feedback (OFB) mode encryption



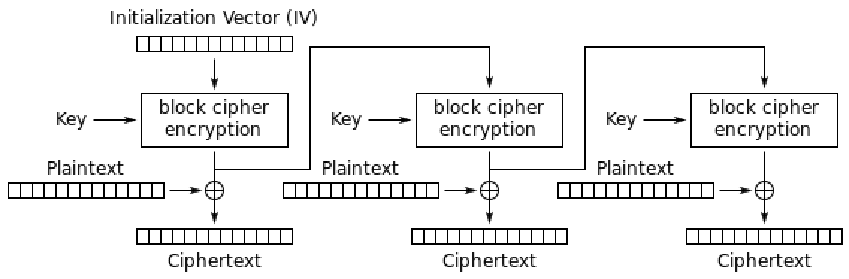
Output Feedback (OFB) mode decryption

Each output feedback block cipher operation depends on all previous ones, and so cannot be performed in parallel

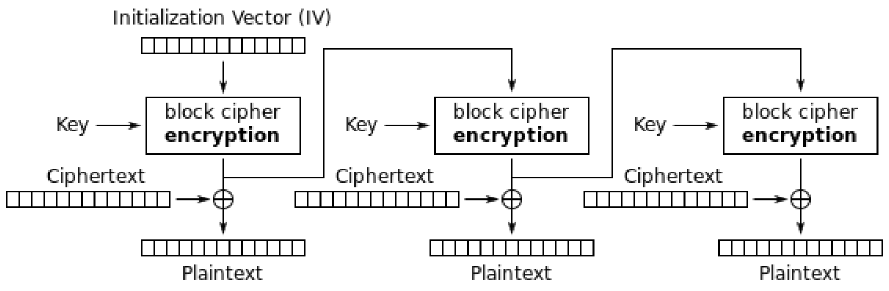
Encryption parallelizable:	No
Decryption parallelizable:	No
Random read access:	No

6. Counter (CTR)

The Output Feedback (OFB) mode makes a block cipher into a synchronous stream cipher. It generates keystream blocks, which are then XORed with the plaintext blocks to get the cipher text. Just as with other stream ciphers, flipping a bit in the cipher text produces a flipped bit in the plaintext at the same location.



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

Note that the nonce in this diagram is equivalent to the initialization vector (IV) in the other diagrams. However, if the offset/location information is corrupt, it will be impossible to partially recover such data due to the dependence on byte offset.

Encryption parallelizable:	Yes
Decryption parallelizable:	yes
Random read access:	Yes

CHAPTER 3

METHODOLOGY

a. Disk Encryption at block level.

In this section, we will see one case of disk encryption and what problem exists within it and then how to solve problem.

For case study, we have taken Disk Encryption technology used in Android OS. Android is a mobile operating system developed by Google and has the largest installed base of all operating systems of any kind.

Disk encryption methods aim to provide three distinct properties:

a. The data on the disk should remain confidential.
b. Data read/write should be fast after encryption, no matter where on the disk the data is stored.
c. The encryption method should not waste disk space.

eMMC are divided into blocks. A block is the smallest readable or writable unit which can be addressed, typically size of 4KB. Disk Encryption uses same key for encrypting each blocks with AES algorithm (Symmetric Algorithm). Assume two separate blocks having same plain text, as we use same disk encryption key for all blocks, cipher text on those blocks will be same. This expose to Malleability attack, where plaintext is known to the adversary, it is possible to change every 2nd plaintext block to a value chosen by the attacker, while the blocks in between are changed to random values, which does not ensure integrity of the encrypted data.

b. Disk Encryption System

To avoid compromise on data integrity, Disk Encryption uses block cipher-based modes, such as CBC (Cipher-block chaining), where previous block's cipher text is xored with the current block's plaintext before encryption. Therefore for same plain test in different block, cipher test will be different.

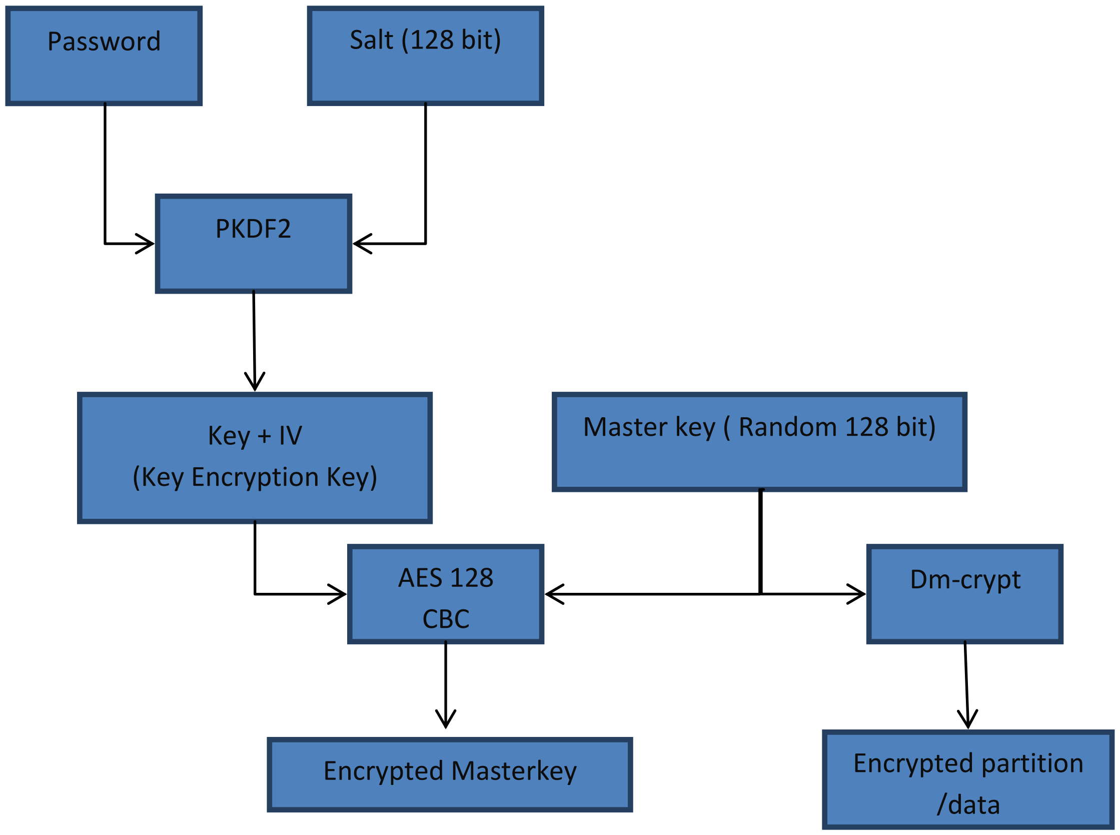
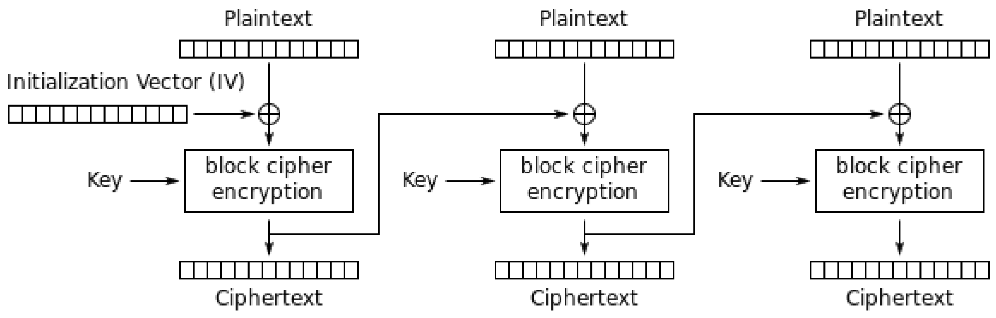


Figure 1: Disk Encryption System on Android

c. Block level encryption of CBC



Cipher Block Chaining (CBC) mode encryption

In case of Cipher Block Chaining, to make each message unique, an initialization vector must be used in the first block. Its main drawbacks are that encryption is sequential (i.e., it cannot be parallelized), and that the message must be padded to a multiple of the cipher block size.

d. Proposed Parallelizable modes of cipher block.

With proposed parallelizable modes of block cipher, we pass block number that we are going to encrypt and keys (supplied by user password) and pass it to HMAC method(keyed-hash message authentication code) to produce a 16 bytes unique random number , that can be used as IV per block. Since IV’s in this case is independent in all blocks, parallelization can be achieve and well suited to operate on a multi-processor machine where blocks can be encrypted in parallel.

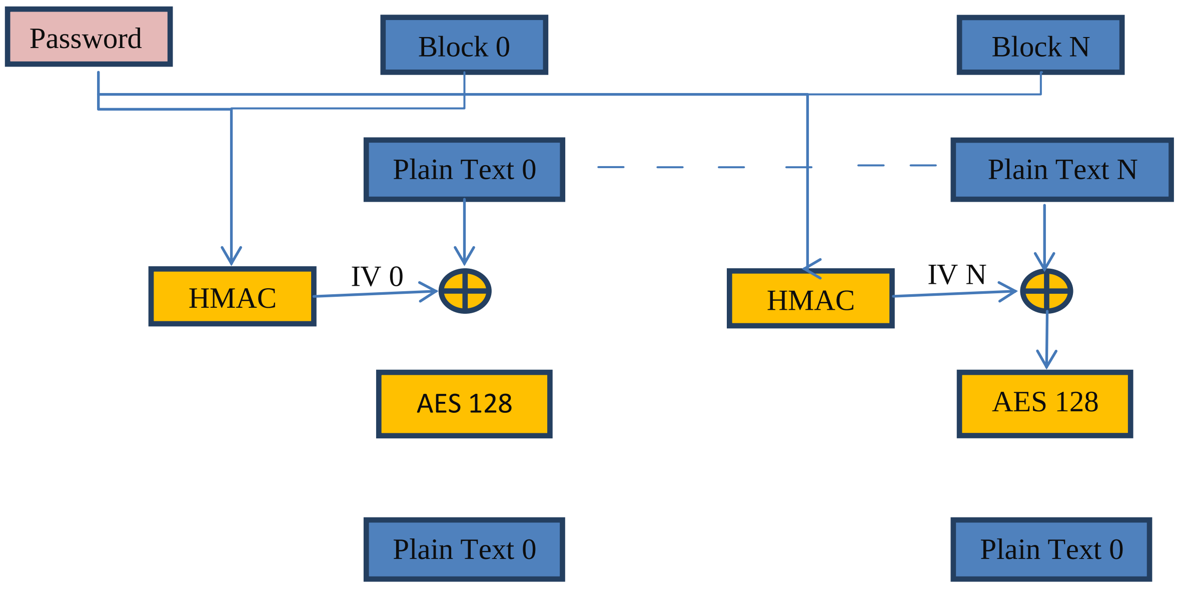


Figure 2: Parallelizable modes of cipher block

e. Proposed Parallelizable modes of cipher block.

Expression for HMAC :

HMAC C(K, m) = H ((K' OR opad) || H ((K' OR ipad) || m))

Where,

H is a cryptographic hash function,

K is the secret key (password in this case)

m is the message to be authenticated (Block Number in this case)

K' is another secret key, derived from the original key K (by padding K to the right with extra zeroes to the input block size of the hash function, or by hashing K if it is longer than that block size),

|| denotes concatenation,

opad is the outer padding (0x5c5c5c…5c5c, one-block-long hexadecimal constant),

ipad is the inner padding (0x363636…3636, one-block-long hexadecimal constant).

f. Encryption Code in CBC mode:

“dtu_mtech_aes_crypt_cbc” method is called for AES encryption with CBC mode.

At the end, method to calculate time of execution also calculated.

```
    start = clock();
        newLen = ReadCurrentBlock(fp1,plain_input, true);  // Read Plain text

        dtu_mtech_aes_setkey_enc( &ctx, key, 256 );  // Prepared key for encryption

        verify_size = CHECK_INTEGRITY_SIZE;

        while(newLen)
        {
                memset(cipher_output, '/0',MAXBUFLEN);

                remainders = dtu_mtech_aes_crypt_cbc( &ctx, false, DTU_MTECH_AES_ENCRYPT, newLen, iv1,
plain_input, cipher_output );   // Call AES Encryption

                if(remainders)
                        newLen+= (16-remainders);
                verify_size-= newLen;
                if(verify_size > 0)
                {
                        WriteCurrentBlock(fp2, cipher_output, newLen);
                }
                newLen = ReadCurrentBlock(fp1,plain_input, true);
        }
        DeinitFS(fp1);
        DeinitFS(fp2);

        end = clock();
   cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;  // Check Execution time for encryption only in CBC
mode.
```

g. Encryption Code in parallelizable mode:

“dtu_mtech_aes_crypt_cbc” method is called for AES encryption with CBC mode.

At the end, method to calculate time of execution also calculated.

```
dtu_mtech_aes_setkey_enc( &ctx, key, 256 );   // Prepared key for encryption

        while(newLen)
        {
                WaitForSingleObject(ghMutex, INFINITE);
                newLen = ReadCurrentBlock(fpgbl,plain_input, true);  // Read Plain text
                MoveNextBlock();
                BN=GetCurrentBlockNumber();  // Get Block size
                ReleaseMutex(ghMutex);
                if(!newLen)
                        break;
                hmac_md5((unsigned char *) &BN, sizeof(BN), (unsigned char *)key, sizeof(key), iv);   // main function to receive Block Number and Keys ( User password) and return 16 bytes random number ( IV per Sector )

                dtu_mtech_aes_crypt_parallelizable( &ctx, true, DTU_MTECH_AES_ENCRYPT, newLen, iv, plain_input, cipher_output );  // Call to custom parallelizable modes of cipher.
        }
```

h. Hash function in parallelizable mode:

```
/*
unsigned char*  text;           Pointer to Block Number Stream
int             text_len;       length of data stream
unsigned char*  key;            pointer to Authentication key ( Password)
int             key_len;        length of Authentication key
unsigned char*  digest;         Per Sector IV
*/

void hmac_md5(const unsigned char *text, int text_len,
                 const unsigned char *key, int key_len,
                 unsigned char *digest)
```

i. Multi Thread to realize parallelization:

Taken 8 thread to run parallel to support encryption process and its well suited to operate on a multi-processor machine where blocks can be encrypted in parallel.

```
#define MAX_THREAD_NO 8


   for (i = 0; i < MAX_THREAD_NO; i++)
      handle[i] = CreateThread(NULL, 0, CryptThread, (LPVOID)i, 0, NULL);

   WaitForMultipleObjects(MAX_THREAD_NO, handle, TRUE, INFINITE);
```

j. Dummy file system

Dummy file system has been used, where large single text files and be read/write is possible. Since File system mount and unmount operation has complex operation and out of scope of proposed solution, we have taken a dummy file system methods to support block wise read and write operation.

```
FILE * initFS(char * path)
{
            mCurrentBlockNumber=0;
```

```
        mWriteBlockNumber = 0;

        FILE *fp = fopen(path, "ab+");
        stat("file.txt", &st);

        steps=0;

        return fp;
}

void DeinitFS(FILE * fp)
{
        fclose(fp);
        fp=0;
        mCurrentBlockNumber=0;
}

int GetCurrentBlockNumber()
{
        return mCurrentBlockNumber;
}




int ReadCurrentBlock(FILE * fp, unsigned char source[MAXBUFLEN], bool  showprogress)
{

        size_t newLen;
        if (fp != NULL)
        {
                newLen = fread(source, sizeof(char), MAXBUFLEN, fp);

                if (newLen == 0)
                {
                        return 0;
                }
                else
                {
                        steps+=newLen;
                        if(showprogress)
                        printProgress( steps /(5244218151));
                }
        }
        return newLen;
}

int WriteCurrentBlock(FILE * fp, unsigned char source[MAXBUFLEN], size_t Len)
{
        size_t newLen=0;
        if (fp != NULL)
        {
                newLen = fwrite((unsigned char*)source, 1, Len, fp);

        }
```

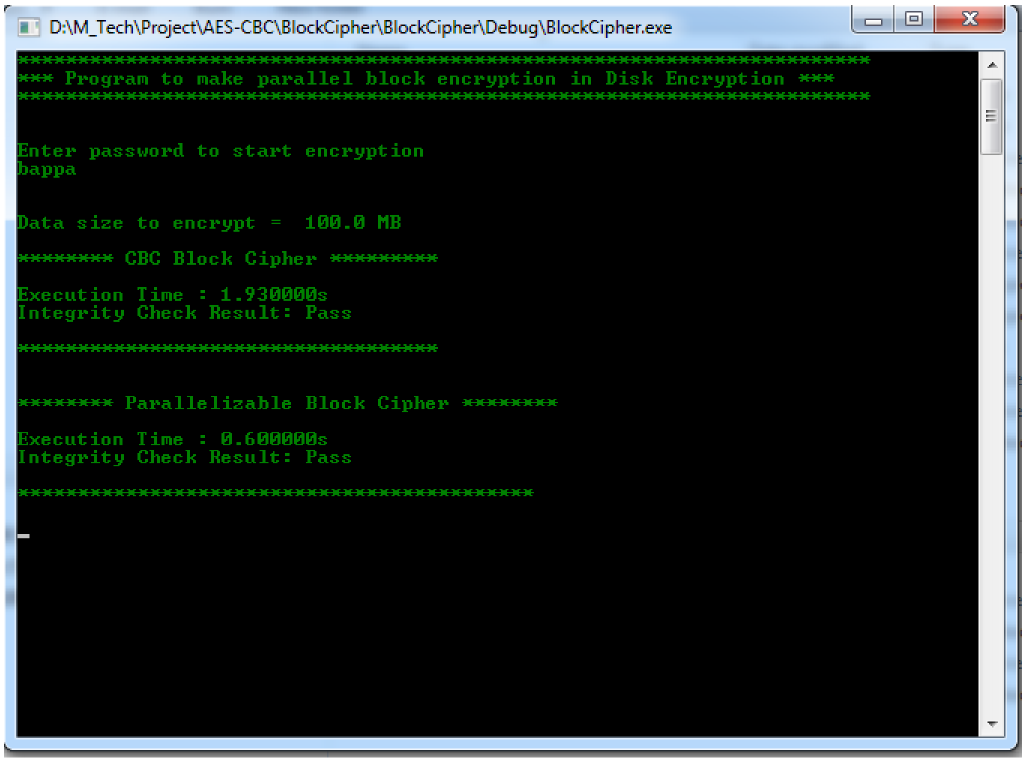
```
                    if (newLen == 0)
                            return -1;
                    else
                            return newLen;

}
```

k. Integrity Checking

Integrity is important property of encryption and should not corrupt data. Data has been encrypted and decrypted for initial 2 blocks and verified if matches with original plain text. Integrity checking block has been made for CBC and proposed parallelizable mode block cipher.

```
check = memcmp  ((char*)plain_input, (char*)plain_output, newLen);
if(0!=check)
 {
        printf("\n\nError: Integrity  fails in CBC mode\n");
 }
```

l. Output to show execution time and integrity check result.



D:\M_Tech\Project\AES-CBC\BlockCipher\BlockCipher\Debug\BlockCipher.exe

```
*************************************************************************
*** Program to make parallel block encryption in Disk Encryption ***
*************************************************************************


Enter password to start encryption
bappa


Data size to encrypt =  100.0 MB

******** CBC Block Cipher *********

Execution Time : 1.930000s
Integrity Check Result: Pass

************************************


******** Parallelizable Block Cipher ********

Execution Time : 0.600000s
Integrity Check Result: Pass

********************************************

_
```

CHAPTER 4

SIMULATION, RESULTS AND DISCUSSION

Disk Encryption and Parallelizable modes of block cipher performance analysis have been done with CBC modes of block cipher. Measurement has been made with amount of data to encrypt and time required completing encryption. With Demo program, we have run and tested it on Intel Core i7 (2.4GHZ) , 8GB of RAM and Windows 7 Enterprise N version.

4.1. Data Comparison of CBC and Proposed Block Cipher.

Program has been tested to establish to evaluate amount of performance can be achieved with parallelization.

Table 1 Performance Evaluation Result

SN	Block Size	Time taken in CBC (Sec)	Time taken in Parallel Block Cipher (Sec)
1	100 MB	6.675	5.01
2	200 MB	13.245	10.28
3	300 MB	19.675	15.595
4	400 MB	31.68	20.4
5	500 MB	34.58	26.085
6	600 MB	39.965	31.1
7	700 MB	46.97	36.607
8	800 MB	53.541	41.371
9	900 MB	64.186	48.796
10	1GB	68.801	52.311
11	2 GB	135.515	107.182
12	3 GB	327.286	154.279
13	4 GB	335.606	220.531
14	5 GB	374.791	269.629

4.2. Graph calculation from performance evaluation

From the table 1, we can see that in case of Parallelizable modes performance has been in terms of time taken to encrypt. It can be observed with below graph that with increase of size of data, Time differences also increases; therefore, it can be ideal solution and can provide significant output for disk encryption for longer disk size for several GB or TB. With above proposed method , we can see average 30% higher encryption speed as compared to CBC.

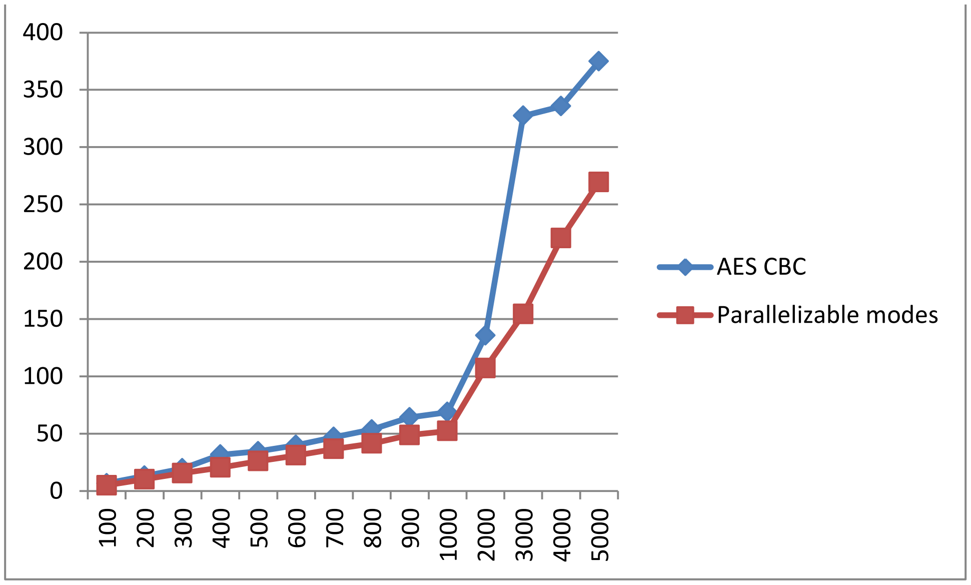


Figure 3 Performance Graph of CBC and Custom penalization mode

CHAPTER 5

CONCLUSION & FUTURE WORK

5.1 CONCLUSION:

In this project, AES and block cipher has been used for disk encryption. For reference, we have taken Disk Encryption methodology from Android Disk Encryption and simulation has been done in Windows 7 machine. With output, we can see approx., 30% higher performance, then CBC block cipher mode.

Approx. 30% higher performance in average and large data size and bigger block size (1 ~ 4 KB) will give best results for disk encryption.

To verify the encryption ability of the proposed method, simulation is done to do encryption along with CBC. The results thus obtained from the simulation show that the proposed method used for disk encryption, which proposes the use of independent IV per block.

5.2 Future work

This may form the future work on Disk Encryption and block cipher. The method to be followed for Disk Encryption is as follows. Bigger encryption blocks size (Currently used 512 bytes) to give encryption results. Here, 512 bytes encryption block has been taken as reference used in Android Disk Encryption. For bigger encryption block, there will be a single IV for single block.

Therefore, there could be security hole for water marking attacks. Other side, for small encryption block, need to generate IV for small block means more IV, therefore, more HMAC call, where more HMAC is expensive method in terms of execution time.

REFERENCES

[1] Like Chen, Runtong Zhang, Runtong Zhang : A Fast Encryption Mode for Block Cipher with Integrity Authentication

[2] Razvi Doomun*, Jayramsingh Doma, Sundeep Tengur : AES-CBC Software Execution Optimization

[3] M.Vaidehi, Dr. B.Justus Rabi : Design and Analysis of AES-CBC Mode for High Security Applications

[4] Zhaohui Wang, Rahul Murmuria, Angelos Stavrou : Implementing and Optimizing an Encryption Filesystem on Android.

[5] Akshay Desai, Krishna Ankalgi, Harish Yamanur, Siddalingesh S. Navalgund: Parallelization of AES algorithm for Disk Encryption Using CBC and ICBC modes

[6] Peter Gutmann, David Naccache, Charles C. Palmer : XTS: A Mode of AES for Encrypting Hard Disks

[7] Cryptography library with source code by MBED TLS https://tls.mbed.org/

[8] Android Disk Encryption process and latest updates https://source.android.com/security/encryption/

[9] Blogs for Anroid Security and Disk Encryption by Nelenkov http://nelenkov.blogspot.in/2014/10/revisiting-android-disk-encryption.html

[10] Blogs for Android Disk Encryption and Various attack by Bappa Mondal

https://bappamondalblog.wordpress/2015/12/26/android-disk-encryption