Nat. Volatiles & Essent. Oils, 2021; 8(6): 4680-4688

NVEO
Natural Volatiles &
Essential Oils

Solutions For Encrypting The Final Block In A Variable Input Message In The Block Cipher Modes Of Operation

Kannan Balasubramanian[1] , K.R.Sekar

[1,2]SASTRA University, India

Abstract

Block cipher Modes of Operation for symmetric ciphers have been introduced to overcome the fixed size of the block ciphers. The objective of this work is to examine methods for encrypting the final block of a variable length message when block cipher modes of operation are used. Symmetric key ciphers like DES and AES encrypt data in fixed block sizes of 64 bits or 128 bits. When the Input data is not a multiple of the block size used by the encryption algorithm, the decryption algorithm cannot recover the plaintext since the plaintext input is not a complete block to the encryption algorithm. To solve this problem several approaches have been used including padding of the last block and the use of elastic block cipher design. This paper proposes including the number of bits as an additional parameter to the encryption procedure which is used to select the number of bits in the ciphe r text block during decryption. The two common approaches for encrypting the final block have their disadvantages, The padding approach when encrypted provides no information about the number of bits to be recovered during decryption. The elastic block cipher approach on the other hand performs encryption modifying the design of the encryption algorithm. The proposed approach does not modify the encryption procedure, but instead proposes to use the number of bits in the final block as a parameter to the encryption algorithm. By using the number of bits in the last block as a parameter, both the decryption module can recover the correct number of bits in the plaintext when decrypted. This method does not need any change to the internal implementation of the encryption algorithm and can correctly decrypt text and binary data when input to the encryption and decryption algorithm. We propose another hybrid method where block cipher encryption is used for all the blocks except the final one and a stream cipher encryption method is used for the final block. The use of random number generation algorithms for use as Initial Vector in the Block Cipher Modes of Operation to provide additional security to the algorithm is also discussed.

Keywords: Symmetric Key Ciphers, Block Cipher Modes of Operation, Block Cipher, Elastic Block Cipher, Random Number Generation, Initial Vector

1. INTRODUCTION

Block cipher modes of operation have been designed for encrypting variable length messages using block ciphers like DES,AES and other symmetric key encryption algorithms that encrypt fixed length plaintexts that are 64 bytes or 128 bytes. Typically, if a text file is encrypted the size of the file will not be a multiple of the block size used by the encrypion algorithm. The last block will be less than the block size used by the encryption algorithm. Various methods have been proposed to handle this problem including the elastic block cipher design proposed by [1]. The design of the block cipher requires that a block encryption requires the specification of the parameter y which is the number of bits in the last block. This parameter can be set to zero for all the blocks except the last block. However the solution to the above problem can be made simpler if we assume that the input

messages are binary contents of text files which occur in multiples of 8 bits. For example, if we assume a 64-bit block cipher, each block contains 8 bytes of data and the last block can contain 1,2 ..or 7 bytes of data. One solution is to put all zeroes after the last byte which is one of the padding methods used by most encryption algorithms. But the encryption algorithm will treat the zeroes also as input and produce an encrypted output and there may not be an easy way to recognise the number of bytes in the last block except by decrypting and noting the padded zeroes in the decrypted output. For many types of inputs such as object code or other binary files , the padded values cannot be easily recognized after decryption since the padded values can be takes as part of the input. This problem does not occur when the plaintext blocks are encrypted since the length of the input message is known. In this paper, a solution for implementing the block cipher mode of operation when the last block is incomplete (i.e. less than the full block size) is described and compared with other similar approaches to this problem. Also we discuss how the initial vector may be randomly generated so the block cipher modes of operation to produce different cipher text values after decryption for the same key.

2. RELATED WORK

Most of the work on Block Cipher Modes of Operation have not considered problem of encrypting the last block in a variable input message. The research work in [8] presents a new block cipher mode of operation by integrating the counter mode of operation with the Cipher Block Chaining Mode of operation. The work presented in [3], the authors describe the attacks on the padding on the ISO CBC mode of encryption. The research presented in [4] proposes to integrate the counter mode with all the modes of operation. The work presented in [5] proposes to constrct stream ciphers from block ciphers and presents the analysis of their security. The work in [6] analyses the performance of the five block cipher modes of operation for the AES encryption algorithm and discusses a new block cipher mode called the Offset Codebook (OCB) mode which addresses the performance degradation problem in the five modes of operation. The research work presented in [7] introduces the new block cipher Offset Codebook mode. The work presented in [8] discusses the performance of the CBC mode on SSL and TLS protocols**.**

3. PROBLEM DESCRIPTION

Both Symmetric key algorithms like DES (Data Encryption Standard) and AES (Advanced Encryption Standard) and Public key algorithms like RSA operate on fixed length input blocks. The DES algorithm uses a 64-bit block size while the AES uses a block size of 128-bits. Typically when a text message coded in ASCII format are encrypted, the message size will not be a multiple of 64 or 128 bits. The last block in the message will be less than 64 bits if DES is used or less than 128 bits if AES is used. The block cipher modes of operation specified for DES or AES [9] do not take into account that the last block of the input message may be less than the full block size and assume that the input message can be split into integral number of blocks specified by the Encryption algorithm. The elastic ciphers were invented to address this problem of the encryption of the incomplete last block. The elastic block cipher design proposed a construction for encrypting a block that is one-bit less than twice the block size., i.e. it proposed to encrypt a block of size b+y where b is the original block size and $0 \leq y < b$ is the number of bits in the last block which can be a maximum of b-1 bits. Using this design encrypting a variable length message can be done by encrypting all the blocks except the last two blocks with y as 0and the last encryption can have b+y bits. This approach has two disadvantages. Firstly, each block encryption needs to specify the y value which is set to 0 for all

but the last encryption. The second disadvantage is that for text encryption the input will have integral number of bytes, hence the last block can be specified using certain number of bytes rather than in bits.

As opposed to the elastic block cipher construction, the padding methods focus on adding padding bytes to the last block[10]. Although the padded bytes can be easily recognised, certain object files like the files with the .EXE file extension will contain the characters in the padding making the padded bytes indistinguishable with real data bytes. To overcome this problem, the proposed solution specifies the number of bytes or bits in the last block when using the block cipher modes of operation. The Table 1 lists the padding methods that can be used in the padding of the last block [11][12]:

Table. 1 Padding Options

S.No	Padding Method	Description
1.	PKCS5 padding	Pad with bytes all of the same value as number of padding bytes
2.	One and Zeroes Padding	Pad with 0x80 followed by values of 0x00
3.	Pad with Zeroes with last byte containing the number of padded bytes	The last byte contains the number of padded bytes and the remaining remaining padded bytes all have zeroes.
4.	Pad with zeroes	All the padded bytes contain the value zero.
5.	Pad with spaces	All the padded bytes contain the space character.
6.	Random Padding	All the padded bytes contain some random value.

4. PROPOSED SOLUTION FOR ENCRYPTION OF FINAL BLOCK.

The proposed solution for the block cipher modes of encryption consists of specifying the number of bits in the last block when the input is provided to the encryption module. The parameters for the block encrypt procedure are as follows:

BLOCK_CIPHER_MODE_ENCRYPT(STARTING_ADDRESS,
NO_OF_BLOCKS,NO_OF_BITS_IN_THE_LAST_BLOCK)

Since the the number of bits in the last block is specified, any method can be used to pad the remaining bits in the last block. The cipher text produced will not be able to recognize the number of bits in the last block. Likewise decryption needs to specify the number of bits in the last block in the encrypted. This information is not available to the decryption module based on the cipher text only, but needs to be stored along with the cipher text to be used while decrypting. The decrypt procedure is specified in a similar manner.

BLOCK_CIPHER_MODE_DECRYPT(STARTING_ADDRESS,NO_OF_BLOCKS,
NO_OF_BITS_IN_THE_LAST_BLOCK).

The block cipher module that encrypts in blocks of certain size can be used to encrypt using any type of block cipher mode of operation. The higher level module that passes the last block for encryption stores the number of valid bits in the last block. The module that encrypts is invoked with the plaintext block and the key as follows:

ENCRYPT(Plaintext_block, Key)

Similarly the module that decrypts can be invoked with cipher text block as follows:

DECRYPT (Cipher text_block, Key)

The issues in the implementation of the block cipher modes of operation are discussed in the following section.

5. COMPARISON WITH ELASTIC BLOCK CIPHER DESIGN

The elastic block cipher design [1][14] was invented to provide a solution to the incomplete final block. The elastic cipher was designed to encrypt b+y bits where b is the block size and y is the number of bits in the last block with $0 \leq y<b$. The implementation of the Block cipher mode of operation can be done in two different ways: In the first method, all the blocks except the last block and the previous block are encrypted with y value set to zero. The last block and the previous block are encrypted with b+y value where y is the number of bits in the last block. As can be seen, this method requires a minimum of one full block and an incomplete block for encryption. There is no way to encrypt a message that is less than one block size. In the second method, the input message is encrypted in 2b bits except for the last encryption which is of sizeb+y where y is less than b. In this method also, there is no way to encrypt a message that is less than one block size in length. The elastic block cipher design required changing the internal implementation of the encryption algorithm for the symmetric ciphers with different block sizes. Since many implementations of the symmetric ciphers are available in software and hardware [15] with the standard block sizes, this change in the encryption algorithm will require implementation of the encryption algorithm with different block sizes may not be a practical approach[16].

6. IMPLEMENTING THE BOCK CIPHER MODES OF OPERATION

There are 5 modes of operation defined for the Data Encryption Standard (DES). They are the Electronic Codebook Mode (ECB), Cipher Block Chaining (CBC), Cipher Feedback Mode (CFB), Output Feedback Mode (OFB) and the counter Mode (CTR) modes of operation[13]. The five modes of operation can be implemented by making use of the simple block encrypt and decrypt function. Encryption of variable length messages can be done by reading a text file, applying padding using one of the padding methods and encrypting the padded block. This method works correctly as long as we are using ASCII data contained in text file. This method will fail if the data to be encrypted is an object file for example, an .EXE file. Implementing the block cipher modes has to be extended by specifying the number of bits in the last block.

The code in Java in Fig.1 implements the Encryption Modes using Cipher Output Stream procedure in Java. The code shown in Fig.2 implements the decryption in Java.

```
void encrypt(String content, String fileName) {
cipher.init(Cipher.ENCRYPT_MODE, secretKey);
byte[] iv = cipher.getIV();

try (FileOutputStreamfileOut = new FileOutputStream(fileName);
CipherOutputStreamcipherOut = new CipherOutputStream(fileOut, cipher)) {
fileOut.write(iv);
cipherOut.write(content.getBytes());}}
```

Fig .1 Encrypting using block cipher modes of operation in Java

The encrypt Modes used in the above algorithm are ECB,CBC,CFB,OFB and CTR modes. Similarly, the decryption modes can be implemented using the Cipher Input Stream procedure in Java.

To implement the encryption of the last block, the encrypt and decrypt and functions should be modified to as follows:

ENCRYPT(plain text_block, Key, No_of_bits_in_the_block)

The Encrypt function applies random padding bits in the block if the number of bits in the block is not the full blocksize and encrypts the block. For decryption, the full ciphertext block has to be considered and not just the number of bits specified.

Likewise, the decrypt function is modified as follows:

DECRYPT(ciphe rtext_block, key, No_of_bits_in_the_block)

The decrypt function decrypts the entire block for decryption and not just the number of bits specified in the argument. After getting the decrypted texr, the desired number of bits are returned to the calling procedure. This general version of the ENCRYPT and DECRYPT procedures can be used to encrypt all the blocks with number of bits set to the block size to encrypt all the blocks except the final block.

```
public static String decrypt(String strToDecrypt, String secret)
   {
try
      {
setKey(secret);
         Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
cipher.init(Cipher.DECRYPT_MODE, secretKey);
return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
      }
catch (Exception e)
      {
System.out.println("Error while decrypting: " + e.toString());
      }
return null;
```

Fig. 2 Decrypting using Block Cipher Modes of Operation in Java

7. A HYBRID METHOD FOR ENCRYPTING VARIABLE LENGTH INPUT MESSAGES

The encryption of the last block becomes an important issue when files containing other than ASCII text such as image files, object code, audio and video data. For text files padding with spaces can be used and the decryption will yield the original message since the spaces can be ignored. For other types of messages, a hybrid method consisting of a block cipher encryption for all the blocks except the final block and a stream cipher method using XOR can be used. Since the block size is either 64 bits (DES) or 128 bits (AES), a simple encryption for the final block using XOR with the key bits can be used. The XOR provides confidentiality and at the same time prevents the need to pass the number of bits to the block cipher routine. The high level encryption program that invokes the block cipher routine decides if it needs to a regular encryption for all blocks except the final block or a simple XOR using the key bits if it is the final block.

9.Use of Initial Vector in the Block Cipher Modes of Operation

The CBC, CFB and OFB modes of operation use an initial vector and chain the outputs from each block. The cipher text produced will depend upon the initial vector chosen. A random initial vector will result in a cip hert ext that is completely unpredictable. Both C++ and Java have built-in functions for choosing random numbers which can be used as initial Vector in the above modes of operation. In C++, the rand() function can be used to get a random number but this function will always return the same sequence of random numbers. To get a different sequence of random numbers, the rand() can be seeded by callingsr and() function.

To prevent the generation of the same sequence of random numbers, the sr and() function can can be seeded by making use of the system time. The code in Fig.3 illustrates how random numbers can be produced using rand() and sr and() functions.

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

int main() {

srand(time(NULL));
cout<<  rand() %100;
}
```

Fig. 3. Generation of random numbers for Initial Vector in C++

1. Using the java. util. random class
2. Math. random method
3. Thread Local Random class

The use of random numbers in the choice of initial vector will ensure that the cipher text produced will produce different cipher texts on different invocations of the encryption procedure with the use of the same key. Although random values as initial vectors can produce different cipher texts for the same, plain text, during decryption the same initial vector should be used to obtain the original plaintext.

The use of initial Vector has to be done with caution when the block cipher modes of operation are used to construct hash functions. For example, one particular construction for hash function uses the CBC mode and uses the final 64-bit block or 128-bit block as the hash value. Since hash functions are used for verification of message integrity, the same hash value needs to be produced for a given message. The use of different initial vectors for a given message will not produce the hash value making the verification invalid. Hence if hash functions are constructed using Block cipher Modes of operation, we need to ensure a fixed initial vector is provided to the encryption function which will ensure that the hash values will be the same when the same message is provided to the hash function,

9. CONCLUSION

Most of the symmetric encryption ciphers in use encrypt in block sizes of 64 bits or 128 bits. For variable length messages, block cipher modes are used. The use of block cipher modes of encryption for symmetric ciphers has to deal with the problem of the incomplete block when encrypting variable size messages. Most of the encyrptions have adopted padding the last block during the block cipher mode of operation. Although padding works well for text messages and ASCII data , it will not work for object files and other binary data. Another method is the elastic block cipher design which focusses on encrypting b+y bits where b is the block size and y varies from zero to b-1 bits. The elastic block cipher design has to encrypta minimum of one block and requires change in the design of the encryption algorithm. As opposed to the elastic block cipher design, another method is proposed where the block encryption module takes the number of bits in the last block as input. By specifying the number of bits in the encrypted data, the calling module can encrypt

variable length data using the block cipher modes of operation, and during decryption, the correct number of bits can be returned to the calling module. Finally, the choice of initial vector by using random number generators in the CBC,CFB and the OFB modes of operation is discussed to provide encryption to produce different cipher texts using the same key.

REFERENCES

1.Cook , D., ,Yung, M.,, and Keromytis, A.D., Elastic Block Ciphers, Cryptology ePrint archive, Report 2004/128,2004.

2. El-Semary, A.M., and Azim, M.M.A, "Counter Chain: A New Block Cipher Mode of Operation" Journal of Information Processing Systems, Vol.11, No.2., pp 266-279, June 2015.

3.Paterson, K.G., and Yau, A.,"Padding Oracle Attacks on the ISO CBC Mode Encryption Standard", Proceedings of the The Cryptographers'Track at the RSA Conference, San Francisco, February 23-27, 2004.

4. Suod, A.T., and Sagheer,A.M., "Counter Mode Development for Block Cipher Operations", Journal of University of Anbar for pure Science, 4(1), 2010.

5. Hudde, H.C., "Building Stream ciphers from Block Ciphers and their Security", http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.392.9064&rep=rep1&type=pdf, Feb 2009.

6. Altigani,A., Abdelmagid,M., and Barry,B., "Analyzing the Performance of Advanced Encryption Standard Block Cipher Modes of Operation: Highlighting the National Institute of Standards and Technology Recommendations", Indian Journal of Science and Technology, 9(28), DOI: 10.17485/ijst/2016/v9i28/97795, July 2016

7. Rogaway,P., Bellare, M., and Black,J., "OCB: A Block-Cipher Mode for Efficient Authenticated Encryption", ACM Transactions in Information and System Security, 6(3), Aug 2oo3, pp 365-403.

8. Kurokawa,T., Nojima,R., and Moriai,S., "On the Security of CBC Mode SSL3.0 and TLS1.0", Journal of Internet Services and System Security, 6(1), Feb 2016, pp 2-19.

9. Sehrawat,D., and Gill,N.S., "Lightweight Block ciphers for IoT based Applications: A Review", International Journal of Applied Engineering Research, 2018,13(5)pp. 2258-2270.

10. Crypto-it. "Block Cipher Modes of Operation", http://www.crypto-it.net/eng/theory/modes-of-block-ciphers.html

11..DI Management, "Using Padding in Encryption", https://www.di-mgt.com.au/cryptopad.html

12. Cryptosys, PKI Pro, Padding Schemes for Block Ciphers, https://www.cryptosys.net/pki/manpki/pki_paddingschemes.html

13. Rogaway Phillip, Evaluation of Some Block Cipher Modes of Operation, carried out for the Cryptography Research and Evaluation Committee(CRYPTREC) , Japan.

14.More,S.P, and Ghorpade, V.R., "Implementation of Elastic AES ". International Journal of Innovations in Engineering Research and Technology, 202, 1-3.

15. Heys, H.M.,, Tutorial on the Implementation of Block Ciphers: Software and Hardware Applications. , Cryptologye Print Archive, Report, 2020/1545, 2020.

16.Granboulan,L, and Pornin, T,."Perfect Block ciphers with Small Blocks", Proceedings of the 14 th international Conference on Fast Software Encryption, Fse, 07., March 2007, pp 452-465.