Extreme Fast AES Library v2.0

1 Introduction

The Advanced Encryption Standard(AES),also known as Rijndael, is a block cipher adopted as an encryption standard by the U.S government.

The Extreme Fast AES Library is based on the official document http://www.csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

It provides the easy to use APIs and fast performance with 128/192/256 bits key length.

2 Cipher Block Modes

There are different block modes you can choose, which are ECB,CBC,PCBC,CFB,OFB and CRT modes. Some of them will need initial vector or an extra feedback size parameter.

2.1 Electronic Codebook (ECB)

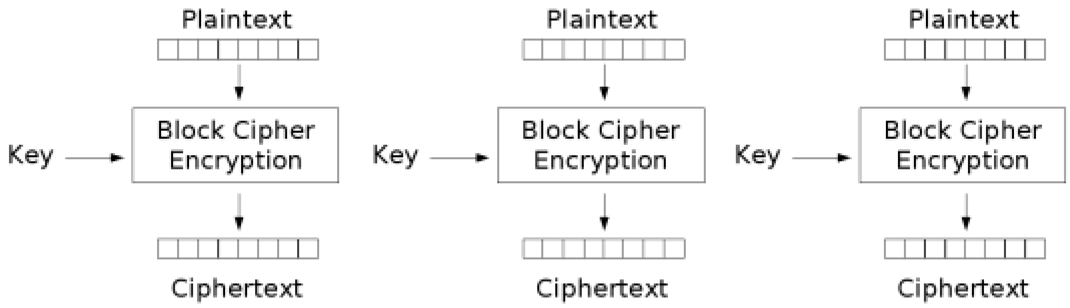

Figure 1. Electronic Codebook mode encryption

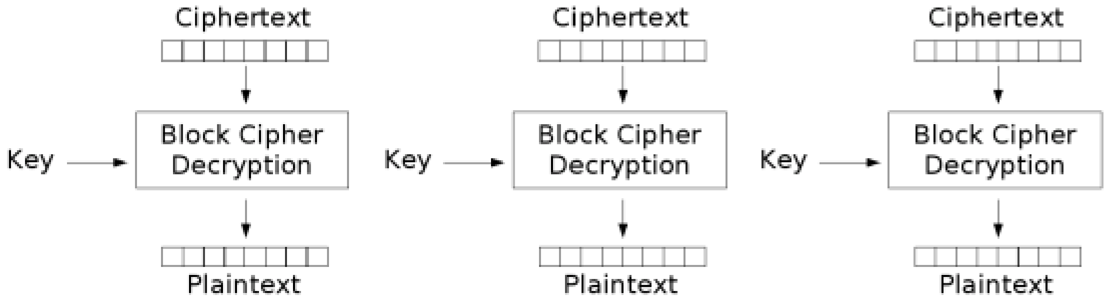

Figure 2. Electronic Codebook mode decryption

2.2 Cipher-block chaining (CBC)

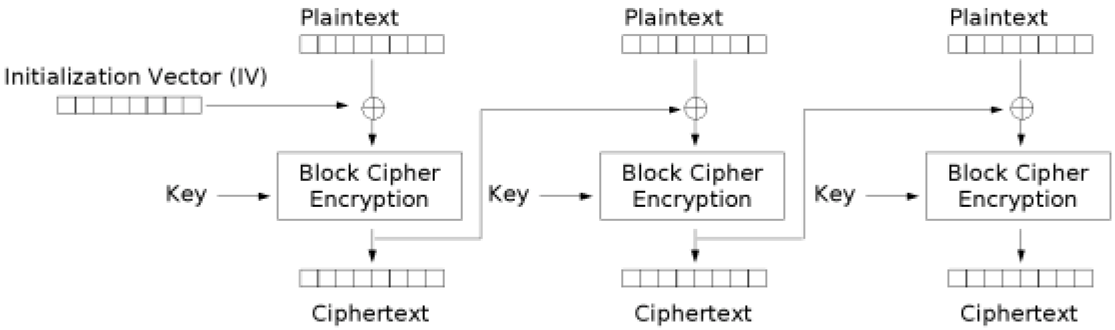

Figure 3. Cipher Block Chaining mode encryption

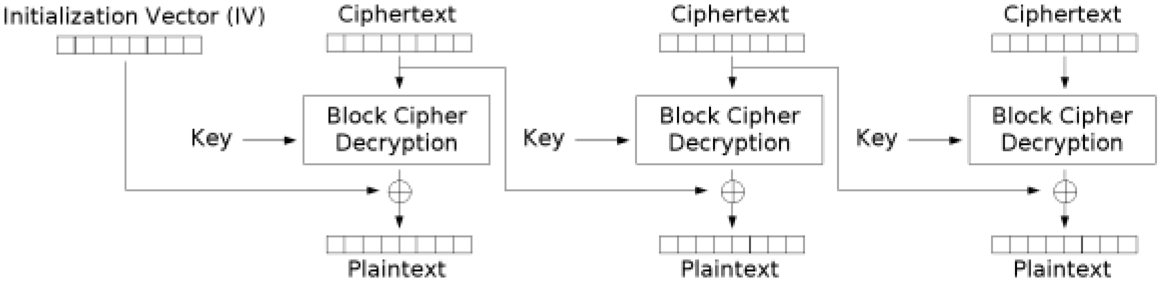

Figure 4. Cipher Block Chaining mode decryption

2.3 Propagating cipher-block chaining (PCBC)

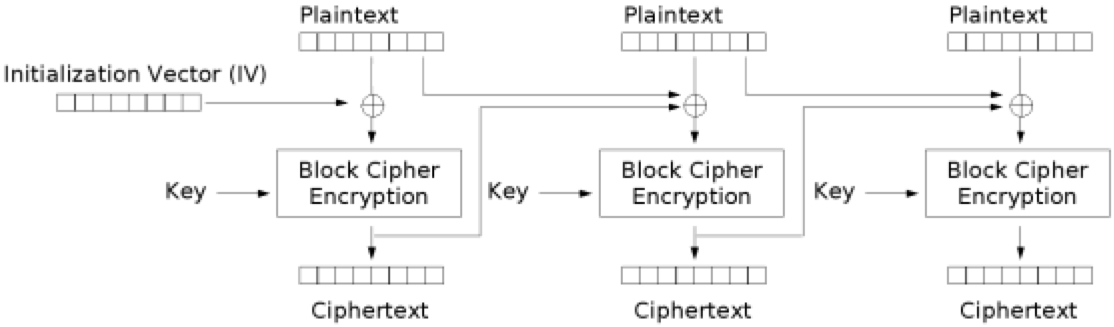

Figure 5. Propagating Cipher Block Chaining mode encryption

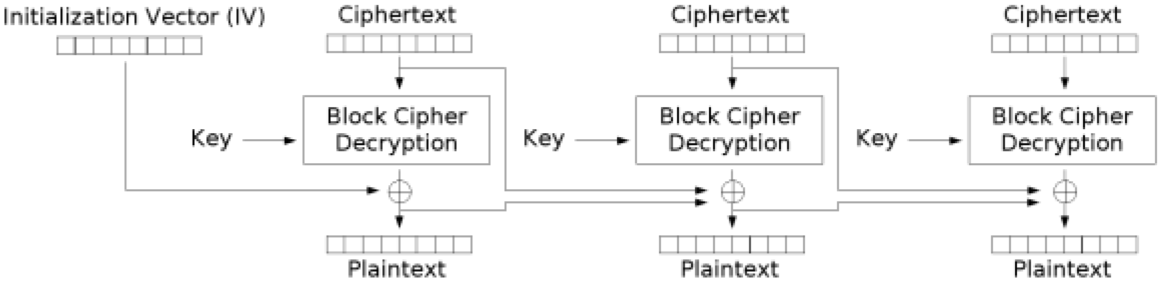

Figure 6. Propagating Cipher Block Chaining mode decryption

2.4 Cipher-Feedback (CFB)

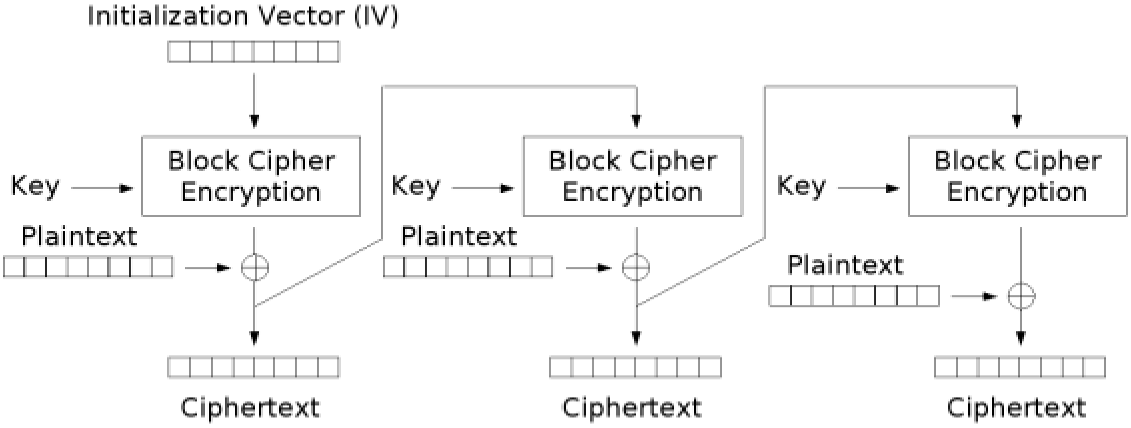

Figure 7. Cipher Feedback mode encryption

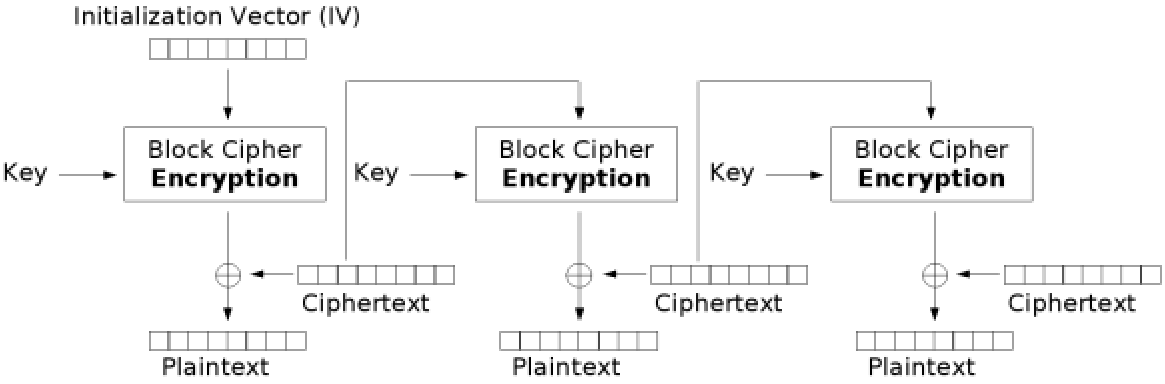

Figure 8. Cipher Feedback mode decryption

2.5 Output Feedback (OFB)

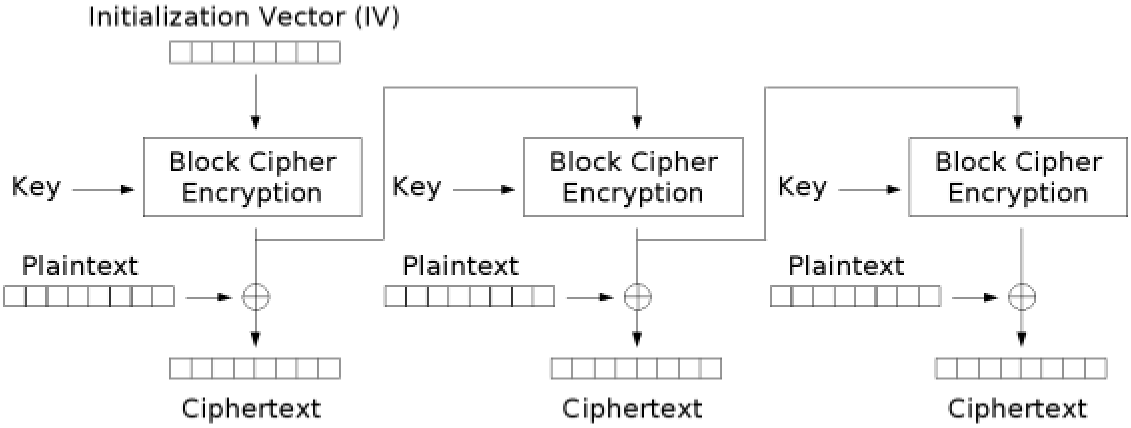

Figure 9. Output Feedback mode encryption

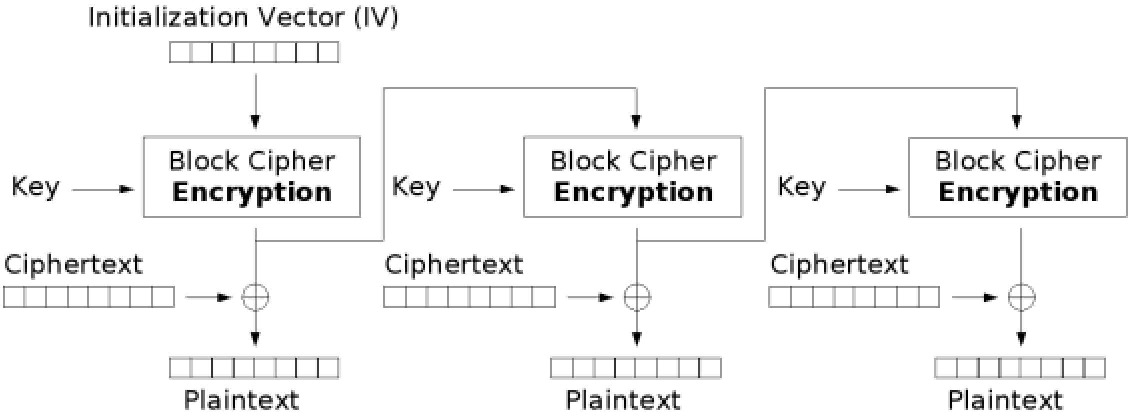

Figure 10. Output Feedback mode decryption

2.6 Counter (CRT)

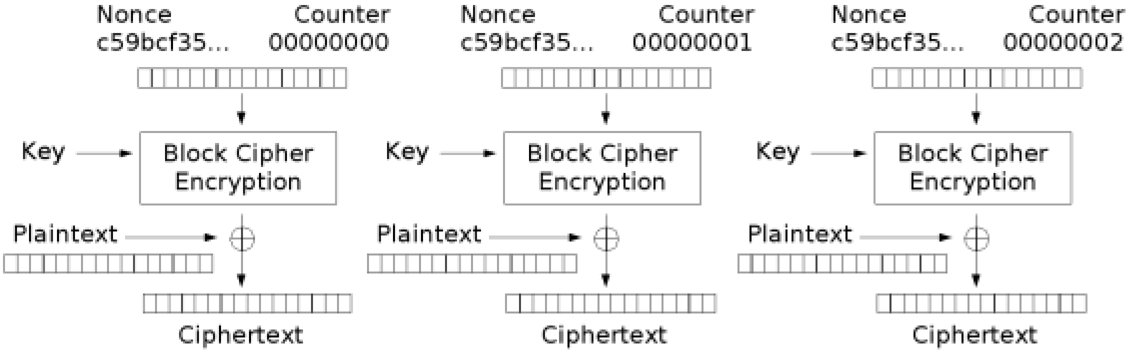

Figure 11. Counter mode encryption

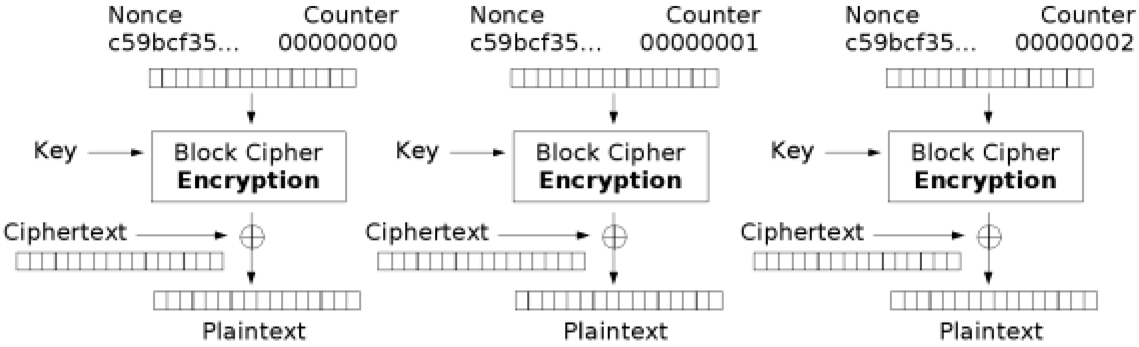

Figure 12. Counter mode decryption

2.7 Block Modes Compare

	Encode/decode with same process	Need initial vector	Chain process
ECB			
CBC		ü	ü
PCBC		ü	ü
CFB	ü	ü	ü
OFB	ü	ü	ü
CRT	ü	ü	ü

Figure 13. Block mode compare

3 Support APIs

3.1 AesSetKey

prototype	void AesSetKey(AesCtx * pContext,int AesKeyLength,int iBlockMode,void * pKey,void * pInitialVector)	
parameters	pContext	Pointer to session context
	AesKeyLength	AES Key length #define AES_KEY_128BIT 0 #define AES_KEY_192BIT 1 #define AES_KEY_256BIT 2
	iBlockMode	Cypher block mode choice,the parameter mainly is because some block mode doesn't need decrypt direction key (CFB,OFB,CRT).
	pKey	Pointer to 128 bit key
	pInitialVector	Pointer to 128 bit initial vector
return	None	
comment	AesSetKey set the encrypt/decrypt key and initial vector for later process. The context design mainly is for thread safe issue. You can pass NULL to initial vector for ECB mode , or you can pass NULL if you want the vector to be zero.	

	AesSetKey will set the default feedback size as 16 bytes. The block modes except the ECB mode will always change the vector within the session. To reset the session, you need to call AesSetKey or AesSetInitVector.

3.2 AesSetInitVector

prototype	AesSetInitVector(AesCtx * pContext, void * pInitialVector)	
parameters	pContext	Pointer to session context
	pInitialVector	Pointer to 128 bit initial vector
return	none	
comment	Set initial Vector for the context. This is only used if you want to reset the session without change the key.	

3.3 AesSetFeedbackSize

prototype	void AesSetFeedbackSize(AesCtx * pContext , int iFeedbackSize)	
parameters	pContext	Pointer to session context
	iFeedbackSize	Feedback size , range from 1 to 16 (Default 16)
return	none	
comment	The feedback size is only used for CFB mode. You can set feedback size to 1 for streaming cipher.	

3.4 AesRoundSize

prototype	int AesRoundSize(int iSize, int iRoundSize)	
parameters	iSize	The input size
	iRoundSize	Usually you will put 16 here,or feedback size when in CFB mode
return	The round size	
comment	This is an utility function to return the round size of iRoundSize, for example , AesRoundSize(18 ,16) will return 32. This can be useful if you want to know the actual output size when encrypt size is not a multiply of 16 (or not a multiply of feedback size in CFB mode) .	

3.5AesEncryptECB，AesDecryptECB

prototype	AesEncryptECB(AesCtx * pContext,void * pDst,void * pSrc,int iSize)
	AesDecryptECB(AesCtx * pContext,void * pDst,void * pSrc,int iSize)	

parameters	pContext	Pointer to session context
	pDst	Pointer to destionation address
	pSrc	Pointer to source data address
	iSize	The source data size
return	none	
comment	The Encrypt/Decrypt in ECB mode. ECB mode is the most simple block cipher mode. It operate each block individually. It will also do the zero padding with 16 bytes alignment.	

3.6 AesEncryptCBC,AesDecryptCBC

prototype	AesEncryptCBC(AesCtx * pContext,void * pDst,void * pSrc,int iSize)
	AesDecryptCBC(AesCtx * pContext,void * pDst,void * pSrc,int iSize)	

parameters	pContext	Pointer to session context
	pDst	Pointer to destionation address
	pSrc	Pointer to source data address
	iSize	The source data size
return	none	
comment	The Encrypt/Decrypt in CBC mode	

3.7 AesEncryptPCBC,AesDecryptPCBC

prototype	AesEncryptPCBC(AesCtx * pContext,void * pDst,void * pSrc,int iSize)
	AesDecryptPCBC(AesCtx * pContext,void * pDst,void * pSrc,int iSize)	

parameters	pContext	Pointer to session context
	pDst	Pointer to destionation address
	pSrc	Pointer to source data address

	iSize	The source data size
return	none	
comment	The Encrypt/Decrypt in PCBC mode	

3.8 AesEncryptOFB,AesDecryptOFB

prototype	AesEncryptOFB(AesCtx * pContext,void * pDst,void * pSrc,int iSize)
	AesDecryptOFB(AesCtx * pContext,void * pDst,void * pSrc,int iSize)	

parameters	pContext	Pointer to session context
	pDst	Pointer to destionation address
	pSrc	Pointer to source data address
	iSize	The source data size
return	none	
comment	The Encrypt/Decrypt in OFB mode	

3.9 AesEncryptCFB,AesDecryptCFB

prototype	AesEncryptCFB (AesCtx * pContext,void * pDst,void * pSrc,int iSize) AesDecryptCFB(AesCtx * pContext,void * pDst,void * pSrc,int iSize)	
parameters	pContext	Pointer to session context
	pDst	Pointer to destionation address
	pSrc	Pointer to source data address
	iSize	The source data size
return	none	
comment	The Encrypt/Decrypt in CFB mode. CFB mode take extra parameter, feedback size. That means it will treat feedback size as block size. That is important to take care about the block size and buffer size. Please take a look in the CFB sample code. The default feedback size is 16 bytes.	

3.10 AesEncryptCRT,AesDecryptCRT

prototype	AesEncryptCRT(AesCtx * pContext,void * pDst,void * pSrc,int iSize)
	AesDecryptCRT(AesCtx * pContext,void * pDst,void * pSrc,int iSize)	

parameters	pContext	Pointer to session context
	pDst	Pointer to destionation address
	pSrc	Pointer to source data address
	iSize	The source data size
return	none	
comment	The Encrypt/Decrypt in CRT mode,CRT mode is popular in network domain. Because it only use encrypt direction，the hardware cost is lower.	

4 Performance bench

Here are the test results in my Pentium4 2.8GHz computer with 10 Mega bytes input size. The time unit is millisecond. (Get by GetTimeTick system call)

	Encrypt	Decrypt	Encrypt In place	Decrypt In place
ECB	73	93	93	78
CBC	78	109	94	94
PCBC	100	93	94	93
OFB (128bits)	78	94	94	78
CFB (128bits)	94	109	93	94
CRT	109	94	109	94

Figure 14. Performance bench

5 Sample Code

Here are some examples to show you how to encode and decode from a file. Please be careful about the buffer handling when you are dealing with The Encrypt/Decrypt in CFB mode with feedback size.

You can always do an in place encrypt/decrypt (source and target address are the same) for these APIs.

5.1 Encrypt a file

```
#include “EfAes.h”
#include <fcntl.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc , char * argv[])
{
     unsigned char key[16]={
                              0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,
                              0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88
                             };
     unsigned char vector[16]={
                                0x1f,0x32,0x43,0x51,0x56,0x98,0xaf,0xed,
                                0xab,0xc8,0x21,0x45,0x63,0x72,0xac,0xfc
                              };
     unsigned char buff[4096];
     int rd_fd,wr_fd, rdsz;
     AesCtx context;
     AesSetKey( &context , AES_KEY_128BIT,BLOCKMODE_CRT, key , vector );

     rd_fd = open(“test.dat”, O_RDONLY);
```

```
    wr_fd = open(“test.encoded”,O_WRONLY | O_CREAT);
    setmode(rd_fd,O_BINARY);
    setmode(wr_fd,O_BINARY);
    while( (rdsz = read(rd_fd, buff ,4096)) > 0 )
    {
        // before last block , the block size should always be the multiply of 16
        // the last block should be handled if the size is not a multiply of 16
        AesEncryptCRT(&context , buff, buff, rdsz );
        rdsz = AesRoundSize( rdsz, 16);
        write( wr_fd , buff , rdsz );
    }
    close(rd_fd);
    close(wr_fd);
}
```

5.2 Decrypt from a file

```
#include “EfAes.h”
#include <fcntl.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc , char * argv[])
{
    unsigned char key[16]={
                                0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,
                                0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88
                               };
    unsigned char vector[16]={
                                  0x1f,0x32,0x43,0x51,0x56,0x98,0xaf,0xed,
                                  0xab,0xc8,0x21,0x45,0x63,0x72,0xac,0xfc
                                };
    unsigned char buff[4096];
    int rd_fd,wr_fd,rdsz;
    AesCtx context;

    AesSetKey( &context , AES_KEY_128BIT, BLOCKMODE_CRT, key , vector );
    rd_fd = open(“test.encoded”, O_RDONLY);
    wr_fd = open(“test.decrypted”,O_WRONLY | O_CREAT);
    setmode(rd_fd,O_BINARY);
    setmode(wr_fd,O_BINARY);
    while( (rdsz = read(rd_fd, buff,4096)) > 0 )
    {
        // the block size should always be the multiply of 16 in decrypt case
        AesDecryptCRT(&context , buff, buff, rdsz );
        write( wr_fd , buff, rdsz );
    }
    close(rd_fd);
    close(wr_fd);
}
```

5.3 Encrypt a file by CFB mode

```
#include “EfAes.h”
#include <fcntl.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>

#define BUFSIZE   4096
int main(int argc , char * argv[])
{
    unsigned char key[16]={ 0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,
                            0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88 };
    unsigned char vector[16]={ 0x1f,0x32,0x43,0x51,0x56,0x98,0xaf,0xed,
                               0xab,0xc8,0x21,0x45,0x63,0x72,0xac,0xfc };
    unsigned char buff[BUFSIZE + AES_PADDING]; // add the AES_PADDING for safe
    int rd_fd,wr_fd,rdsz;
    int iFeedBackSize = 5; // you can change the feedback size from 1 to 16

    // the process block size should be a multiply of feedback size
    int iBlockSize = AesRoundSize(BUFSIZE, iFeedBackSize);
    AesCtx context;
    AesSetKey( &context , AES_KEY_128BIT, BLOCKMODE_CFB, key , vector );
    AesSetFeedbackSize( &context , iFeedBackSize);

    rd_fd = open(“test.dat”, O_RDONLY);
    wr_fd = open(“test.encoded”,O_WRONLY | O_CREAT);
    setmode(rd_fd,O_BINARY);
    setmode(wr_fd,O_BINARY);
    while( (rdsz = read(rd_fd, buff, iBlockSize)) > 0 )
    {
        AesEncryptCFB(&context , buff, buff, rdsz );
        //  the output size should always be the multiply of Feedback Size
        rdsz = AesRoundSize( rdsz, iFeedBackSize);
        write( wr_fd , buff, rdsz );
    }
    close(rd_fd);
    close(wr_fd);
}
```

5.4 Decrypt a file by CFB mode

```
#include “EfAes.h”
#include <fcntl.h>
#include <io.h>
#include <stdio.h>
#include <stdlib.h>

#define BUFSIZE   4096
int main(int argc , char * argv[])
{
    unsigned char key[16]={ 0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88,
                             0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88 };
    unsigned char vector[16]={ 0x1f,0x32,0x43,0x51,0x56,0x98,0xaf,0xed,
                                0xab,0xc8,0x21,0x45,0x63,0x72,0xac,0xfc };
    unsigned char buff[BUFSIZE + AES_PADDING]; // add the AES_PADDING for safe
    int rd_fd,wr_fd,rdsz;
    int iFeedBackSize = 5; // you can change the feedback size from 1 to 16
    // the process block size should be a multiply of feedback size
    int iBlockSize = AesRoundSize(BUFSIZE, iFeedBackSize);

    AesCtx context;
    AesSetKey( &context , AES_KEY_128BIT, BLOCKMODE_CFB, key , vector );
    AesSetFeedbackSize( &context , iFeedBackSize);

    rd_fd = open("test.encoded", O_RDONLY);
    wr_fd = open("test.decrypted",O_WRONLY | O_CREAT);
    setmode(rd_fd,O_BINARY);
    setmode(wr_fd,O_BINARY);
    while( (rdsz = read(rd_fd, buff, iBlockSize)) > 0 )
    {
        AesDecryptCFB(&context , buff, buff, rdsz );
        //  the output size should always be the multiply of Feedback Size
        rdsz = AesRoundSize( rdsz, iFeedBackSize);
        write( wr_fd , buff, rdsz );
    }
    close(rd_fd);
    close(wr_fd);
}
```