**КУРСОВАЯ РАБОТА**

по дисциплине «Защита Информации»

на тему «Реализация алгоритма AES»

|  |  |
| --- | --- |
| Выполнила  студентка гр. 3530904/90102 | Ли Ицзя |
| Руководитель | [Медведев Б. М.](https://dl.spbstu.ru/course/view.php?id=4608) |

**Оглавление**

[Цель работы 3](#_Toc131411238)

[Обзор Литературы 3](#_Toc131411239)

[История 3](#_Toc131411240)

[Теоретические основы 3](#_Toc131411241)

[Базовая структура AES 4](#_Toc131411242)

[Обзор процесса шифрования AES 5](#_Toc131411243)

[Китайский стандартный алгоритм шифрования – SM4 8](#_Toc131411244)

[Основные операции 9](#_Toc131411245)

[Основная функция 9](#_Toc131411246)

[Раундовая функция 10](#_Toc131411247)

[Характеристики SM4 12](#_Toc131411248)

[Реализация 13](#_Toc131411249)

[AES-128 13](#_Toc131411250)

[Подмена / замена байтов. (SubBytes) 13](#_Toc131411251)

[Циклические сдвиг рядов. (ShiftRows) 15](#_Toc131411252)

[Перемешение столбцов. (MixColumns) 15](#_Toc131411253)

[Суммирование состояния.  (AddRoundKey) 17](#_Toc131411254)

[SM4 18](#_Toc131411255)

[Сравнение AES с SM4 21](#_Toc131411256)

[Результат 23](#_Toc131411257)

[AES-128 24](#_Toc131411258)

[SM4 27](#_Toc131411259)

[Вывод 31](#_Toc131411260)

[Список лителатуры 32](#_Toc131411261)

# Цель работы

Целью данной курсовой работы является изучение технологии защиты информации в AES алгоритме. Реализовать более эффективный способ защиты информации передающейся по алгоритму.

Основные задачи исследования:

1. Изучение принципа работы алгоритм AES;
2. Реализация алгоритм AES;
3. Сравнение данной реализации с другими алгоритмами (алгоритмы симметричного шифрования)

# Обзор Литературы

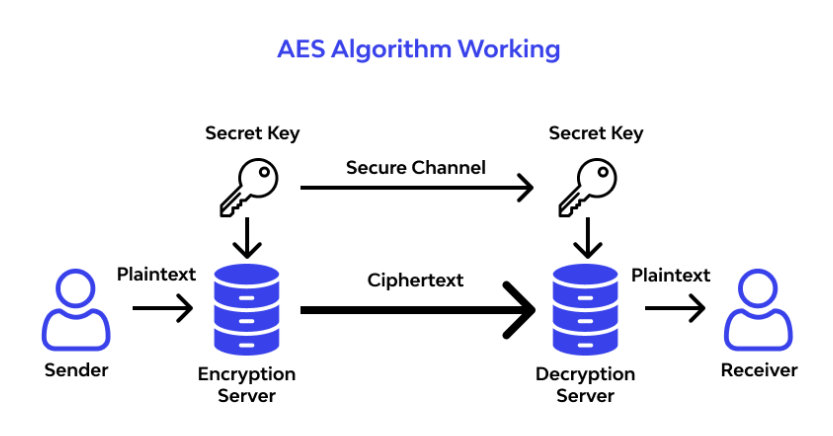
## История

Блочное шифрование — это разновидность криптосистемы, широко используемая в настоящее время. Алгоритм шифрования DES[1][2] (Data Encryption Standard) создал прецедент для общедоступного алгоритма шифрования, так что владелец зашифрованного текста, который понимает алгоритм, но не имеет правильного ключа, не может найти данные открытого текста с помощью алгоритма. Его безопасность зависит от расчета и времени взломать пароль. Однако с развитием компьютерных и сетевых технологий практика показала, что DES не является надежным криптографическим алгоритмом, и его жизнь подошла к концу во всех аспектах. Фактически, DES Challenge, запущенный RSA в 1997 году, доказал, что 56-битный ключ DES слишком короткий. Хотя тройной DES может решить проблему длины ключа, он не очень эффективен, а дизайн DES предназначен в основном для аппаратной реализации[3][4]. В настоящее время во многих областях необходимо реализовать относительно эффективный алгоритм для программного обеспечения. Ввиду этого, на таком фоне и появился алгоритм шифрования AES[5]. Полное название AES - Advance Encryption Standard. Проект был инициирован Американским национальным институтом стандартов и технологий (NIST) в 1997 г. и собирал алгоритмы. В 2000 г. было решено принять Rijndael в качестве окончательного алгоритма. В 2001 г. он был одобрен Министерством торговли США в качестве нового алгоритма федеральный стандарт шифрования информации (FIPS PUB 197). В качестве замены алгоритму шифрования DES алгоритм шифрования AES обладает безопасностью, высокой эффективностью и неизменно хорошей производительностью в различных аппаратных и программных операционных средах, поэтому алгоритм имеет высокий потенциал развития и хорошую практическую ценность.

## Теоретические основы

AES - это симметричный тип шифрования. Алгоритм симметричного шифрования использует один и тот же ключ для шифрования и дешифрования. С другой стороны, асимметричныйключевые системы используют разные ключи для каждогоиз двух процессов: шифрование и дешифрование.

Конкретный процесс шифрования выглядит следующим образом[6]:



Plaintext: незашифрованные данные.

Secret Key: пароль, используемый для шифрования открытого текста.В алгоритме симметричного шифрования ключи шифрования и дешифрования совпадают. Ключ генерируется путем согласования между получателем и отправителем, но он не может быть передан напрямую по сети, иначе произойдет утечка ключа.Обычно ключ шифруется алгоритмом асимметричного шифрования, а затем передается другой стороне через сети, или непосредственно обсуждаемый лицом к лицу ключ. Ключ ни в коем случае нельзя упускать, иначе злоумышленник восстановит зашифрованный текст и украдет конфиденциальные данные.

Ciphertext: данные, обработанные функцией шифрования

### Базовая структура AES

Далее, AES - блочный шифр. Он называется «блочным», потому что этот тип шифра разделяет информацию, которая должна быть зашифрована (известный как открытый текст) на разделы, называемые блоками. В стандарта AES длина пакета может составлять только 128 бит, то есть каждый пакет составляет 16 байт (8 бит на байт). Длина ключа может быть 128 бит, 192 бит или 256 бит. Длина ключа отличается, и рекомендуемое количество раундов шифрования также отличается, как показано в следующей таблице:

|  |  |  |  |
| --- | --- | --- | --- |
| AES | Длина ключа (бит) | Длина блоков(бит) | Количество раундов(бит) |
| AES-128 | 128 | 128 | 10 |
| AES-192 | 192 | 128 | 12 |
| AES-256 | 256 | 128 | 14 |

Здесь реализован AES-128, то есть длина ключа 128 бит, а количество раундов шифрования 10.

В функции шифрования выполняется циклическая функция, и эта циклическая функция выполняется 10 раз. Первые 9 исполнений этой циклической функции одинаковы, и отличается только 10-й раз. То есть блок открытого текста будет зашифрован на 10 раундов. Суть AES заключается в реализации всех операций за один раунд.

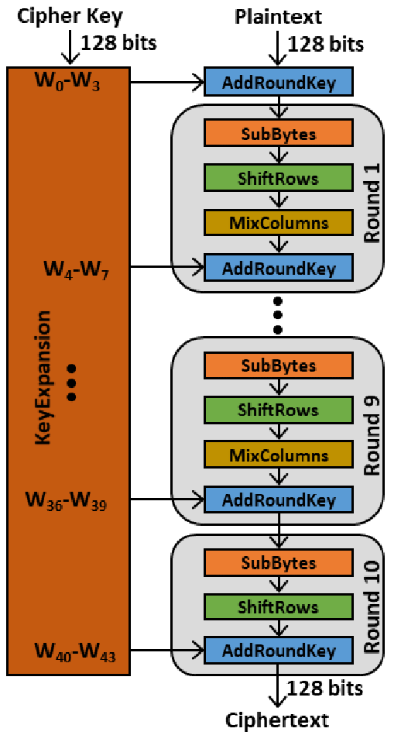
### Обзор процесса шифрования AES

Алгоритм шифрования AES проходит через несколько раундов шифрования. Он может даже пройти 10, 12 или 14 раундов.

Каждый раунд включает в себя те же шаги, что и ниже.

* Подмена / замена байтов. (SubBytes)
* Циклические сдвиг рядов. (ShiftRows)
* Перемешение столбцов. (MixColumns)
* Суммирование состояния.  (AddRoundKey)

После последнего раунда алгоритм пройдет еще один раунд. В этом наборе алгоритм выполнит шаги с 1 по 4. кроме3 шагу.



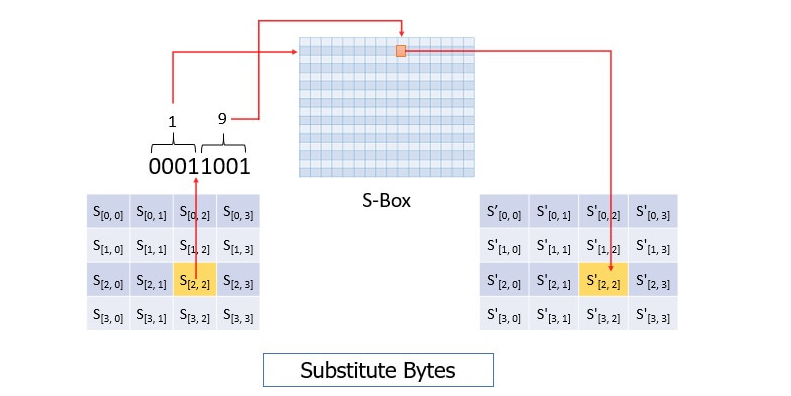
Давайте теперь углубимся в детали.

#### Разделение данных на блоки

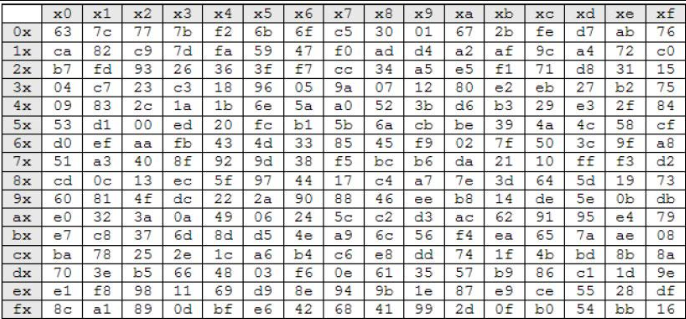
Единицей обработки AES является байт.128-битный входной блок открытого текста A и входной ключ K разделены на 16 байтов, которые записываются как A = a0 a1 ... a15 и K = k0 k1 ... k15 соответственно.

#### Подмена / замена байтов (SubBytes)

Подмена байтов AES на самом деле представляет собой простую операцию поиска в таблице, и ее основная функция заключается в завершении преобразования одного байта в другой через блок S[5]. AES определяет S-блок и обратный S-блок.



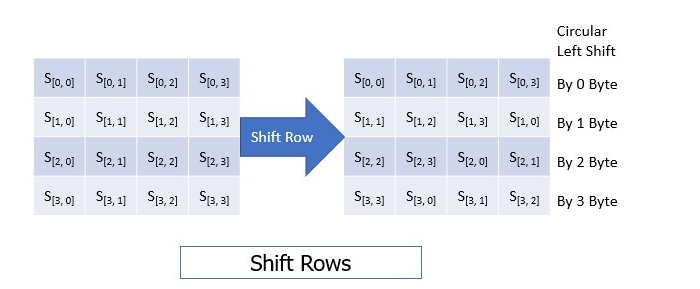
S-Блок:



Элементы в матрице состояния отображаются в новый байт следующим образом: берем старшие 4 бита байта как значение строки, а младшие 4 бита — как значение столбца, и вынимаем элементы соответствующей строки в блок S или обратный блок S в качестве вывода.

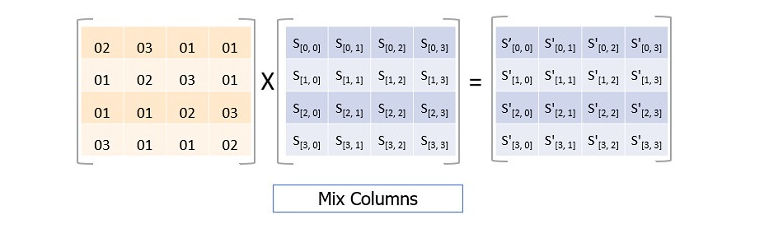
#### Циклические сдвиги строк на разное число позиций (ShiftRows)

Сдвиг строки — это простая операция кругового сдвига влево, которая выполняет перестановку между байтами в матрице 4x4. При длине ключа 128 бит 0-я строка матрицы состояния сдвигается влево на 0 байт, первая строка сдвигается влево на 1 байт, вторая строка сдвигается влево на 2 байта, а третья строка сдвинута влево на 3 байта, как показано на рисунке ниже. Показано как ниже:



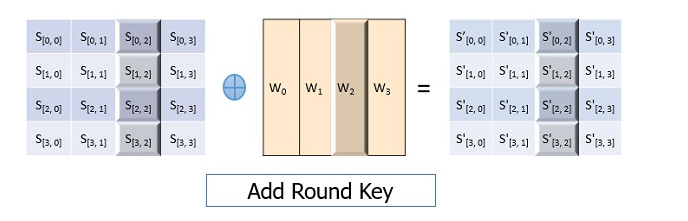
#### Перемешение столбцов (MixColumns)

Преобразование смешивания столбцов реализуется путем умножения матриц. Матрица состояния после сдвига строки умножается на фиксированную матрицу для получения матрицы запутанного состояния, как показано на следующем рисунке:



#### Суммирование состояния (AddRoundKey)

Добавление ключа раунда заключается в выполнении побитовой операции XOR[5] над 128-битным ключом раунда Ki и данными в матрице состояний.



#### Зашифрование

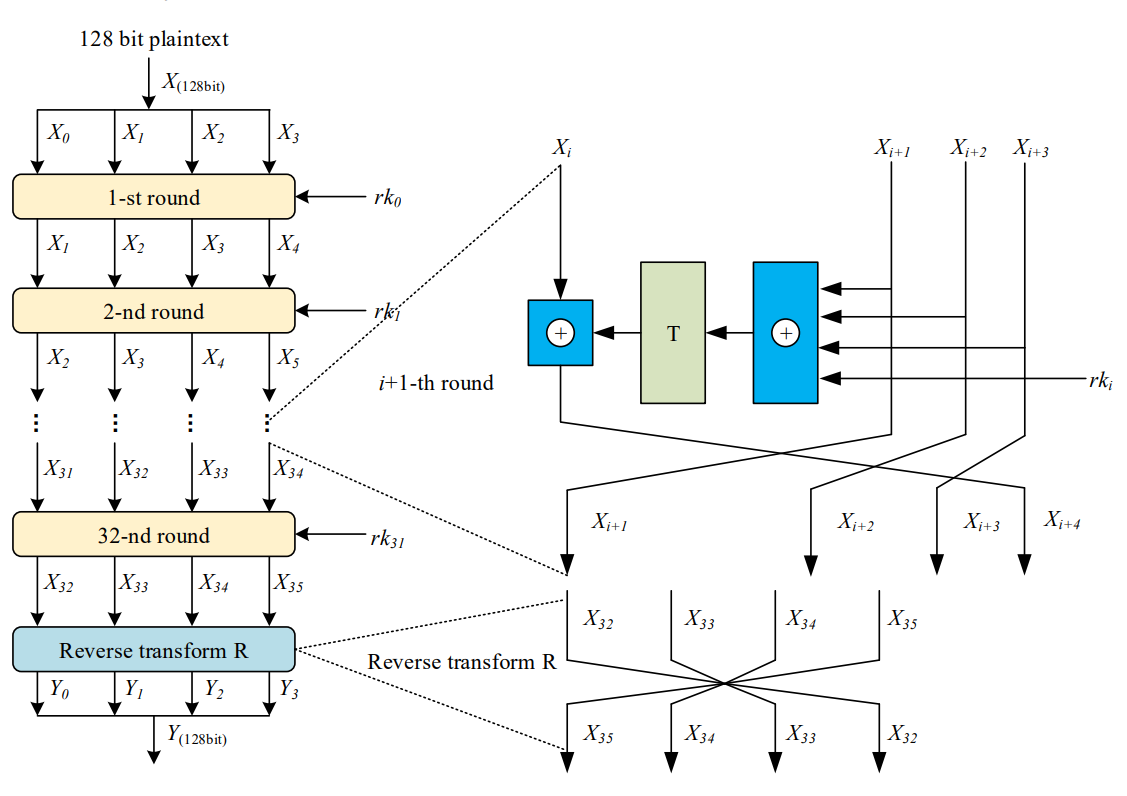
Процесс шифрования AES и дешифрования AES одинаков и также начинается с функции добавления круглого ключа. 16-байтовый шифрованный текст в виде матрицы состояния 4 x 4 подвергается операции XOR[5] с уникальным ключом из 4 слов.

Последовательность ключей в шифровании меняется на противоположную во время расшифровки. И все остальные циклические функции также инвертируются в процессе расшифровки, чтобы получить исходный 16-байтовый блок обычного текста.

## Китайский стандартный алгоритм шифрования – SM4

Алгоритм SM — это китайский государственный стантартный алгоритм шифрования, это один мз внутренних алгоритмов шифрования, признанных Государственным управлением криптографии. В основном это SМ1, SМ2, SМ3, SМ4. Среди них SM1 - симметричное шифрование, и его стойкость шифрования эквивалентна AES, но алгоритм не раскрывается. При вызове этого алгоритма его нужно вызывать через интерфейс определенного шифровального чипа. SM2 - асимметричное шифрование алгоритм. SM3 — алгоритм дайджеста информации, SM4 — алгоритм симметричного шифрования.

SM4 — алгоритм блочного шифрования используемый в Китае для беспроводных локальных сетей (WLAN Authentication and Privacy Infrastructure -WAPI). Длина пакета данных составляет 128 бит, а длина ключа составляет 128 бит. Алгоритм шифрования и алгоритм расширения ключей используют 32 раунда итеративной структуры.



### Основные операции

Алгоритм SM4 использует сложение по модулю 2 и циклический сдвиг в качестве основной операции.

### Основная функция

1. S-таблица

S- таблица SM4 представляет собой нелинейное преобразование подстановки в единицах байтов, и его криптографическая функция заключается в запутывании. Вход и выход S- таблица представляют собой 8-битные байты. По сути, это нелинейная перестановка 8 бит. Например, если вход таблицы S равен EF, для замены используется значение на пересечении строки E и столбца F.

1. Нелинейное преобразование τ

Нелинейное преобразование τ SM4 является нелинейным преобразованием в единицах слов. Он состоит из 4 S-блоков, расположенных рядом друг с другом. Нелинейное преобразование здесь такое же, как принцип преобразования S-блока zuc ранее.Слово имеет 4 байта, и каждый байт изменяется с помощью S-блока, а затем результаты соединяются встык. Разница в том, что ZUC использует два S-бокса, а здесь только один S-бокс.

1. Линейное преобразование τ

Линейное преобразование L LL — это компонент линейного преобразования, единицей обработки которого является слово, а его вход и выход — 32-битные слова. Роль его криптографии заключается в том, чтобы играть роль в распространении.

Пусть входом L LL будет слово B, а выходом — слово C, тогда:

1. Синтетическое преобразование T

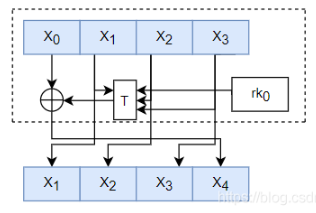
Синтетическое преобразование состоит из нелинейного преобразования τ \tauτ и линейного преобразования L.

Синтетическое преобразование T TT играет роль путаницы и диффузии и повышает безопасность пароля.

### Раундовая функция

Раудолвая функция алгоритма SM4 представляет собой криптографическую функцию со словом в качестве единицы обработки.

Предположим, что вход функции округления F FF равен ( X 0 , X 1 , X 2 , X 3), четыре 32-битных слова, всего 128 бит. Раудовый ключ — rk, который также является 32-битным словом. Его вывод также представляет собой 32-битное слово.



**Процесс шифрования SM4**

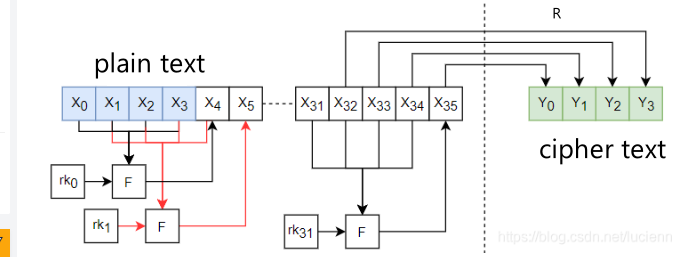
Алгоритм шифрования SM4 является блочным алгоритмом. Длина пакета данных составляет 128 бит, а длина ключа — 128 бит. Алгоритм шифрования использует итеративную структуру из 32 раундов, и в каждом раунде используется ключ раунда.

Пусть входный текст , а исходный ключ , всего 32 слова, выходный зашифрованный текст

Шофрование:

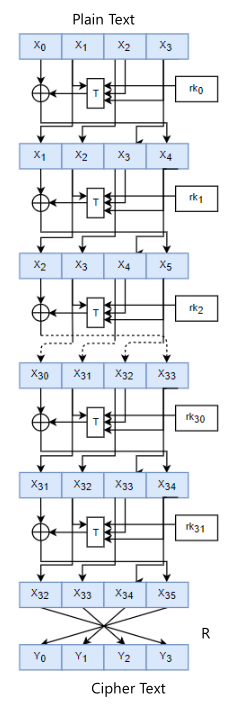
Чтобы соответствовать порядку, требуемому алгоритмом дешифрования, после алгоритма шифрования требуется обработка Rв обратном порядке:

Блок-схема алгоритма шифрования выглядит следующим образом:



Из рисунка видно, что SM4 шифрует 4 слова за раз, чтобы сгенерировать промежуточный шифртекст из одного слова.Этот промежуточный шифртекст, этот промежуточный шифртекст и первые три слова соединяются вместе для следующей обработки, и всего 32 раунда итерации и, наконец, получить зашифрованный текст из четырех слов.

Процесс шифрования также может выглядеть так:



### Характеристики SM4

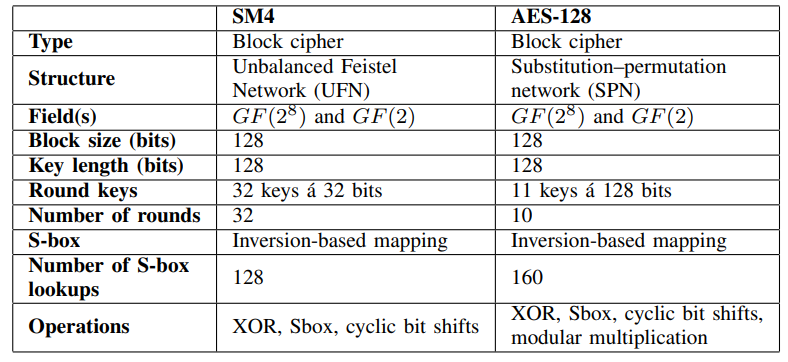
Алгоритм был рассекречен в январе 2006. Несколько характеристик SM4:

* Размер блока составляет 128 бит.
* Используется 8 - битный S-box
* Размер ключа 128 бит.
* Используются только операции типа XOR, кругового сдвига и приложения S-Box
* Выполняется 32 раунда для обработки одного блока
* Каждый раунд обновляет четверть (32 бита) внутреннего состояния.
* Используется не линейное составление ключа (key schedule) для создания раундовых ключей.
* При расшифровке используются те же ключи что и при шифровании, но в обратном порядке.

Алгоритм SM4 был разработан профессором Лю Шу-Ваном (LU Shu-wang). SM4 был предложен как шифр используемый в стандарте IEEE 802.11i[8], но был быстро заменён ISO. Одной из причин этого была оппозиция WAPI fast-track продвигаемая IEEE.

SM4 часто используется для шифрования передачи данных в государственных системах, например, когда мы передаем параметры из внешнего интерфейса в фоновый, мы можем использовать этот алгоритм.

Поскольку длина ключа и размер пакета алгоритма SM4 такие же, как у алгоритма AES, мы часто сравниваем их вместе. Здесь мы также сравним различия в производительности шифрования и дешифрования между алгоритмом SM4 и алгоритмом AES.



# Реализация

Мы будем реализовывать 128-битный алгоритм AES.

## AES-128

typedef bitset<8> byte;

typedef bitset<32> word;

extern byte S[256]; //S-box

extern byte inv\_S[256];

extern word rcon[10]; // Rcon

extern byte encry\_s[4 \* 4];

extern byte decry\_s[4 \* 4];

// длина начального ключа и количество раундов расширения

const int Nr = 10;// Количество раундов

const int Nk = 4; // Начальный ключ состоит из четырех слов.

### Подмена / замена байтов. (SubBytes)

Основная функция слоя подстановки байтов - позволить входным данным завершить отображение из одного байта в другой через таблицу S\_box.Таблица S\_box здесь рассчитывается по определенному методу.Нам нужно только использовать вычисленные результаты S\_box. Таблица S\_box представляет собой массив с 256-байтовыми элементами.Его можно определить как одномерный массив или как двумерный массив 16.16.Если он определен как двумерный массив, данные S\_box можно прочитать Метод заключается в использовании старших четырех битов каждого байта входных данных в качестве первого нижнего индекса, а четвертого бита в качестве второго нижнего индекса. Здесь рекомендуется рассматривать его как одномерный массив. Обратный S-блок соответствует S-блоку и используется для обработки данных при дешифровании.Наша программная обработка при дешифровании называется обратной подстановкой байтов, но при шифровании используется другой блок подстановки.

const unsigned char S\_Table[16][16] =

{

    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,

    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,

    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,

    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,

    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,

    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,

    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,

    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,

    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,

    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,

    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,

    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,

    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,

    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,

    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,

    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16

};

int Plain\_S\_Substitution(unsigned char \*PlainArray)

{

    int ret = 0;

    for (int i = 0; i < 16; i++)

    {

        PlainArray[i] = S\_Table[PlainArray[i] >> 4][PlainArray[i] & 0x0F];

    }

    return ret;

}

//inv S-box

const unsigned char ReS\_Table[16][16] =

{

    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB,

    0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB,

    0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,

    0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,

    0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,

    0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,

    0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,

    0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,

    0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,

    0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,

    0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,

    0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,

    0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,

    0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,

    0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,

    0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D

};

// inv SubBytes

int Cipher\_S\_Substitution(unsigned char \*CipherArray)

{

    int ret = 0;

    for (int i = 0; i < 16; i++)

    {

        CipherArray[i] = ReS\_Table[CipherArray[i] >> 4][CipherArray[i] & 0x0F];

    }

    return ret;

}

### Циклические сдвиг рядов. (ShiftRows)

Операция сдвига строки является самой простой, она используется для обработки входных данных в виде матрицы 4\*4 байт, а затем замены байтов этой матрицы на позиции. Подуровень ShiftRows относится к слою ручной диффузии AES, цель которого состоит в том, чтобы распространить преобразование на один бит, чтобы повлиять на все состояние, чтобы добиться лавинного эффекта. Обработка смещения строки при шифровании противоположна обработке при дешифровании, и мы называем здесь обработку при дешифровании ретроградным смещением. Это называется сдвигом строки, потому что он работает только между строками матрицы 4 \* 4 с 4 байтами данных в строке. При шифровании первую строку матрицы оставить без изменений, вторую строку сдвинуть влево на 8 бит (один байт), третью строку — влево на 2 байта, четвертую строку — влево на 3 байта. Наоборот, при расшифровке первая строка остается неизменной, вторая строка сдвигается вправо на один байт, третья строка сдвигается вправо на 2 байта, а четвертая строка сдвигается вправо на 3 байта. Операция окончена!

int ShiftRows(unsigned int \*PlainArray)

{

    int ret = 0;

     //PlainArray[0] = PlainArray[0];

     PlainArray[1] = (PlainArray[1] >> 8) | (PlainArray[1] << 24);

     PlainArray[2] = (PlainArray[2] >> 16) | (PlainArray[2] << 16);

     PlainArray[3] = (PlainArray[3] >> 24) | (PlainArray[3] << 8);

    return ret;

}

int ReShiftRows(unsigned int \*CipherArray)

{

    int ret = 0;

     //CipherArray[0] = CipherArray[0];

     CipherArray[1] = (CipherArray[1] << 8) | (CipherArray[1] >> 24);

     CipherArray[2] = (CipherArray[2] << 16) | (CipherArray[2] >> 16);

     CipherArray[3] = (CipherArray[3] << 24) | (CipherArray[3] >> 8);

    return ret;

}

### Перемешение столбцов. (MixColumns)

Подуровень запутывания столбца является наиболее сложной частью алгоритма AES и относится к слою диффузии.Операция запутывания столбца является основным элементом диффузии в алгоритме AES.Она смешивает каждый столбец входной матрицы, так что каждый байт входной повлияет на 4 выходных байта. Комбинация подслоя смещения строк и подслоя путаницы столбцов делает возможным, что каждый байт матрицы зависит от 16 байтов открытого текста после трех циклов обработки. Он охватывает умножение матриц, сложение и умножение в полях Галуа.

const unsigned char MixArray[4][4] =

{

    0x02, 0x03, 0x01, 0x01,

    0x01, 0x02, 0x03, 0x01,

    0x01, 0x01, 0x02, 0x03,

    0x03, 0x01, 0x01, 0x02

};

int MixColum(unsigned char(\*PlainArray)[4])

{

    int ret = 0;

    unsigned char ArrayTemp[4][4];

     memcpy(ArrayTemp, PlainArray, 16);

     for (int i = 0; i < 4; i++)

    {

        for (int j = 0; j < 4; j++)

        {

            PlainArray[i][j] =

                MixArray[i][0] \* ArrayTemp[0][j] +

                MixArray[i][1] \* ArrayTemp[1][j] +

                MixArray[i][2] \* ArrayTemp[2][j] +

                MixArray[i][3] \* ArrayTemp[3][j];

        }

    }

    return ret;

}

Мы обнаружили, что в матричном умножении есть операции сложения и умножения.Мы также упомянули, что операция сложения в расширенном поле эквивалентна операции XOR, а операция умножения нуждается в особом способе обработки, поэтому мы сначала поместим код в Замените знак плюс на символ XOR, а затем определите умножение поля Галуа как функцию с двумя параметрами, и пусть он возвращает окончательный результат вычисления. Таким образом, код обфускации столбца будет выглядеть так:

const unsigned char MixArray[4][4] =

{

    0x02, 0x03, 0x01, 0x01,

    0x01, 0x02, 0x03, 0x01,

    0x01, 0x01, 0x02, 0x03,

    0x03, 0x01, 0x01, 0x02

};

int MixColum(unsigned char(\*PlainArray)[4])

{

    int ret = 0;

    unsigned char ArrayTemp[4][4];

     memcpy(ArrayTemp, PlainArray, 16);

     for (int i = 0; i < 4; i++)

    {

        for (int j = 0; j < 4; j++)

        {

            PlainArray[i][j] =

                GaloisMultiplication(MixArray[i][0], ArrayTemp[0][j]) ^

                GaloisMultiplication(MixArray[i][1], ArrayTemp[1][j]) ^

                GaloisMultiplication(MixArray[i][2], ArrayTemp[2][j]) ^

                GaloisMultiplication(MixArray[i][3], ArrayTemp[3][j]);

        }

    }

    return ret;

}

Далее нам нужно иметь дело только с обработкой, связанной с умножением поля Галуа.

// Функция: операция умножения GF(128) в поле Галуа.

char GaloisMultiplication(unsigned char Num\_L, unsigned char Num\_R)

{

    unsigned char Result = 0;

    while (Num\_L)

    {

        if (Num\_L & 0x01)

        {

            Result ^= Num\_R;

        }

        Num\_L = Num\_L >> 1;

        if (Num\_R & 0x80)

        {

            Num\_R = Num\_R << 1;

            Num\_R ^= 0x1B;

        }

        else

        {

            Num\_R = Num\_R << 1;

        }

    }

    return Result;

}

Разница между расшифрованной путаницей в обратном столбце и путаницей в прямом столбце заключается в том, что используемая левая матрица умножения отличается, и это обратная матрица левой матрицы умножения путаницы в прямом столбце, то есть матрица данных остается левой. умноженная После матрицы матрица данных никак не изменится.

### Суммирование состояния.  (AddRoundKey)

На уровне добавления ключей есть два входных параметра: открытый текст и подключ k[0], и оба входных параметра имеют длину 128 бит. k[0] фактически эквивалентен ключу k, конкретная причина вводится при генерации ключа. Как мы упоминали ранее при введении сложения и вычитания в расширенной области, операция сложения и вычитания в расширенной области эквивалентна операции XOR, поэтому обработка здесь чрезвычайно проста.Вам нужно только нажать два входных данных.Байт XOR операция получит результат операции.

int AddRoundKey(unsigned char(\*PlainArray)[4], unsigned char(\*ExtendKeyArray)[44], unsigned int MinCol)

{

    int ret = 0;

    for (int i = 0; i < 4; i++)

    {

        for (int j = 0; j < 4; j++)

        {

            PlainArray[i][j] ^= ExtendKeyArray[i][MinCol + j];

        }

    }

    return ret;

}

## SM4

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

string BinToHex(string str) { //#2->#16

string hex = "";

int temp = 0;

while (str.size() % 4 != 0) {

str = "0" + str;

}

for (int i = 0; i < str.size(); i += 4) {

temp = (str[i] - '0') \* 8 + (str[i + 1] - '0') \* 4 + (str[i + 2] - '0') \* 2 + (str[i + 3] - '0') \* 1;

if (temp < 10) {

hex += to\_string(temp);

}

else {

hex += 'A' + (temp - 10);

}

}

return hex;

}

string HexToBin(string str) {//#16->#2

string bin = "";

string table[16] = { "0000","0001","0010","0011","0100","0101","0110","0111","1000","1001","1010","1011","1100","1101","1110","1111" };

for (int i = 0; i < str.size(); i++) {

if (str[i] >= 'A' && str[i] <= 'F') {

bin += table[str[i] - 'A' + 10];

}

else {

bin += table[str[i] - '0'];

}

}

return bin;

}

int HexToDec(char str) {//#16->#10

int dec = 0;

if (str >= 'A' && str <= 'F') {

dec += (str - 'A' + 10);

}

else {

dec += (str - '0');

}

return dec;

}

string LeftShift(string str, int len) {//shift left

string res = HexToBin(str);

res = res.substr(len) + res.substr(0, len);

return BinToHex(res);

}

string XOR(string str1, string str2) {//XOR

string res1 = HexToBin(str1);

string res2 = HexToBin(str2);

string res = "";

for (int i = 0; i < res1.size(); i++) {

if (res1[i] == res2[i]) {

res += "0";

}

else {

res += "1";

}

}

return BinToHex(res);

}

string NLTransform(string str) {//Non-Linear function t

string Sbox[16][16] = { {"D6","90","E9","FE","CC","E1","3D","B7","16","B6","14","C2","28","FB","2C","05"},

{"2B","67","9A","76","2A","BE","04","C3","AA","44","13","26","49","86","06","99"},

{"9C","42","50","F4","91","EF","98","7A","33","54","0B","43","ED","CF","AC","62"},

{"E4","B3","1C","A9","C9","08","E8","95","80","DF","94","FA","75","8F","3F","A6"},

{"47","07","A7","FC","F3","73","17","BA","83","59","3C","19","E6","85","4F","A8"},

{"68","6B","81","B2","71","64","DA","8B","F8","EB","0F","4B","70","56","9D","35"},

{"1E","24","0E","5E","63","58","D1","A2","25","22","7C","3B","01","21","78","87"},

{"D4","00","46","57","9F","D3","27","52","4C","36","02","E7","A0","C4","C8","9E"},

{"EA","BF","8A","D2","40","C7","38","B5","A3","F7","F2","CE","F9","61","15","A1"},

{"E0","AE","5D","A4","9B","34","1A","55","AD","93","32","30","F5","8C","B1","E3"},

{"1D","F6","E2","2E","82","66","CA","60","C0","29","23","AB","0D","53","4E","6F"},

{"D5","DB","37","45","DE","FD","8E","2F","03","FF","6A","72","6D","6C","5B","51"},

{"8D","1B","AF","92","BB","DD","BC","7F","11","D9","5C","41","1F","10","5A","D8"},

{"0A","C1","31","88","A5","CD","7B","BD","2D","74","D0","12","B8","E5","B4","B0"},

{"89","69","97","4A","0C","96","77","7E","65","B9","F1","09","C5","6E","C6","84"},

{"18","F0","7D","EC","3A","DC","4D","20","79","EE","5F","3E","D7","CB","39","48"} };

string res = "";

for (int i = 0; i < 4; i++) {

res = res + Sbox[HexToDec(str[2 \* i])][HexToDec(str[2 \* i + 1])];

}

return res;

}

string LTransform(string str) {//Linear function L

return XOR(XOR(XOR(XOR(str, LeftShift(str, 2)), LeftShift(str, 10)), LeftShift(str, 18)), LeftShift(str, 24));

}

string L2Transform(string str) {//L'

return XOR(XOR(str, LeftShift(str, 13)), LeftShift(str, 23));

}

string T(string str) {// For encryption

return LTransform(NLTransform(str));

}

string T2(string str) {// For key expansion

return L2Transform(NLTransform(str));

}

string KeyExtension(string MK) {// Key Expansion

string FK[4] = { "A3B1BAC6", "56AA3350", "677D9197", "B27022DC" };

string CK[32] = { "00070E15", "1C232A31", "383F464D", "545B6269",

"70777E85", "8C939AA1", "A8AFB6BD", "C4CBD2D9",

"E0E7EEF5", "FC030A11", "181F262D", "343B4249",

"50575E65", "6C737A81", "888F969D", "A4ABB2B9",

"C0C7CED5", "DCE3EAF1", "F8FF060D", "141B2229",

"30373E45", "4C535A61", "686F767D", "848B9299",

"A0A7AEB5", "BCC3CAD1", "D8DFE6ED", "F4FB0209",

"10171E25", "2C333A41", "484F565D", "646B7279" };

string K[36] = { XOR(MK.substr(0,8),FK[0]),XOR(MK.substr(8,8),FK[1]),XOR(MK.substr(16,8),FK[2]),XOR(MK.substr(24),FK[3]) };

string rks = "";

for (int i = 0; i < 32; i++) {

K[i + 4] = XOR(K[i], T2(XOR(XOR(XOR(K[i + 1], K[i + 2]), K[i + 3]), CK[i])));

rks += K[i + 4];

}

return rks;

}

string encode(string plain, string key) {// Encryption

cout << "Round Key & Output Status: " << endl;

cout << endl;

string cipher[36] = { plain.substr(0,8),plain.substr(8,8),plain.substr(16,8),plain.substr(24) };

string rks = KeyExtension(key);

for (int i = 0; i < 32; i++) {

cipher[i + 4] = XOR(cipher[i], T(XOR(XOR(XOR(cipher[i + 1], cipher[i + 2]), cipher[i + 3]), rks.substr(8 \* i, 8))));

cout << "rk[" + to\_string(i) + "] = " + rks.substr(8 \* i, 8) + " X[" + to\_string(i) + "] = " + cipher[i + 4] << endl;

}

cout << endl;

return cipher[35] + cipher[34] + cipher[33] + cipher[32];

}

string decode(string cipher, string key) {

cout << "Round Key & Output Status: " << endl;

cout << endl;

string plain[36] = { cipher.substr(0,8),cipher.substr(8,8), cipher.substr(16,8), cipher.substr(24,8) };

string rks = KeyExtension(key);

for (int i = 0; i < 32; i++) {

plain[i + 4] = XOR(plain[i], T(XOR(XOR(XOR(plain[i + 1], plain[i + 2]), plain[i + 3]), rks.substr(8 \* (31 - i), 8))));

cout << "rk[" + to\_string(i) + "] = " + rks.substr(8 \* (31 - i), 8) + " X[" + to\_string(i) + "] = " + plain[i + 4] << endl;

}

cout << endl;

return plain[35] + plain[34] + plain[33] + plain[32];

}

int main() {

string str = "0123456789AAAAAAAAAAAB9876543210";

cout << "Plain Text: " << str.substr(0, 8) << " " << str.substr(8, 8) << " " << str.substr(16, 8) << " " << str.substr(24, 8) << endl;

cout << endl;

string key = "0123456789ABCDEFFEDCBA9876543210";

cout << "Encypt with: " << key.substr(0, 8) << " " << key.substr(8, 8) << " " << key.substr(16, 8) << " " << key.substr(24, 8) << endl;

cout << endl;

string cipher = encode(str, key);

cout << "Cipher text: " << cipher.substr(0, 8) << " " << cipher.substr(8, 8) << " " << cipher.substr(16, 8) << " " << cipher.substr(24, 8) << endl;

cout << endl;

cout << "Cipher text: " << cipher.substr(0, 8) << " " << cipher.substr(8, 8) << " " << cipher.substr(16, 8) << " " << cipher.substr(24, 8) << endl;

cout << endl;

cout << "Decrypt with: " << key.substr(0, 8) << " " << key.substr(8, 8) << " " << key.substr(16, 8) << " " << key.substr(24, 8) << endl;

cout << endl;

string plain = decode(cipher, key);

cout << "Got from decrption: " << plain.substr(0, 8) << " " << plain.substr(8, 8) << " " << plain.substr(16, 8) << " " << plain.substr(24, 8) << endl;

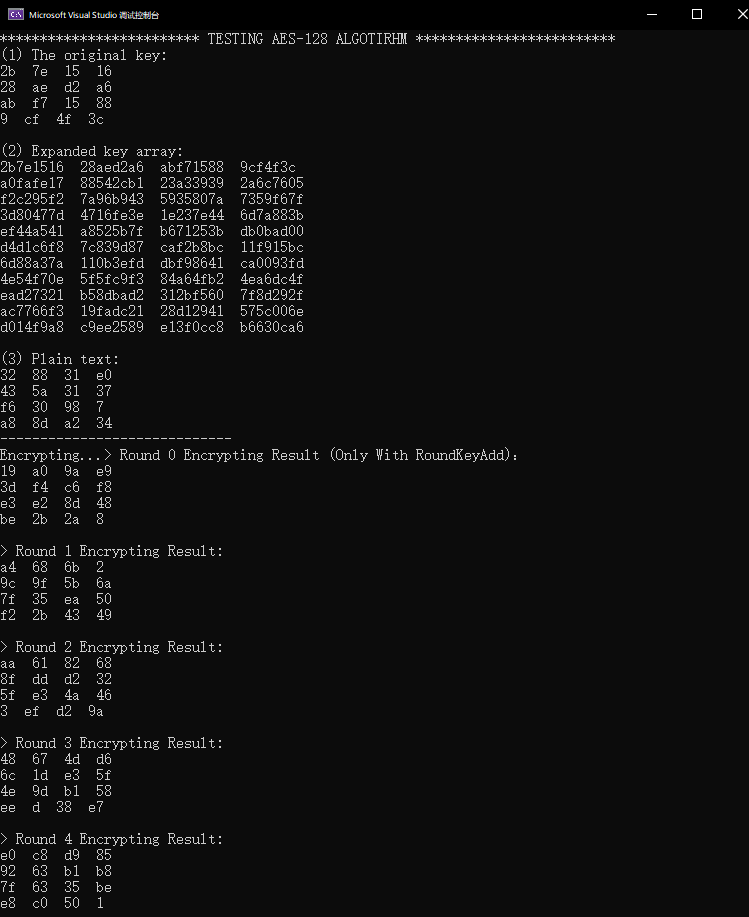
}

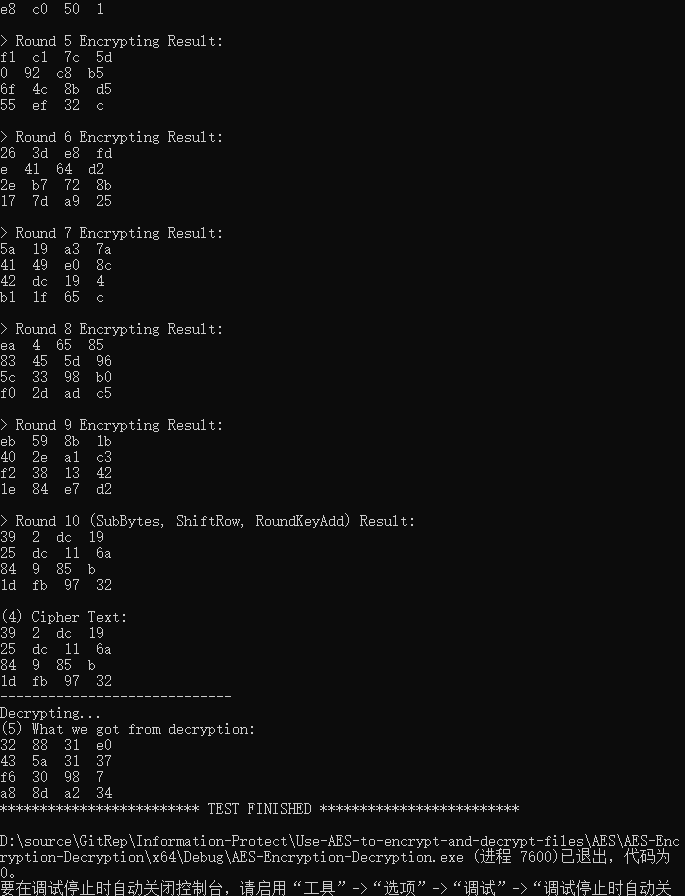
## Сравнение AES с SM4

public class CompareAESandSM4 {  
  
 public static void main(String[] args) {  
 encryDecry();  
 }  
  
 public void encryDecry() throws Exception{  
 SM4StringEncryptor sm4 = new SM4StringEncryptor();  
 String path = "D:\\Users\\xlj\\Downloads\\sylog.txt";  
  
 File file = new File(path);  
 InputStreamReader reader = new InputStreamReader(new FileInputStream(path));  
  
 BufferedReader br = new BufferedReader(reader);  
 StringBuilder fileContent = new StringBuilder();  
  
 String line = "";  
 line = br.readLine();  
 while (line != null) {  
 line = br.readLine();  
 fileContent.append(line);  
 }  
 System.*out*.println("Original File Size:" + fileContent.length() /1024/1024 + "Mb");  
  
 StringBuilder fileContent = new StringBuilder("xlj12442");  
 long startTime = System.*currentTimeMillis*();  
  
 String encryContent= sm4.encrypt(fileContent.toString());  
 System.*out*.println("---After sm4 encrypt:" + encryContent);  
 System.*out*.println("File size after sm4:" + encryContent.length() /1024/1024 + "Mb");  
  
 long endTime = System.*currentTimeMillis*();  
 System.*out*.println("Time cost:" + (endTime - startTime) + "ms");  
  
 String decryContent = sm4.decrypt(encryContent);  
 System.*out*.println("---After sm4 decrypt:" + decryContent);  
 System.*out*.println("File size after sm4 decryption:" + decryContent.length() /1024/1024 + "Mb");  
 long endTime1 = System.*currentTimeMillis*();  
 System.*out*.println("Time cost:" + (endTime1 - endTime) + "ms");  
  
 // aes  
 byte[] key = SecureUtil.generateKey(SymmetricAlgorithm.AES.getValue()).getEncoded();  
 SymmetricCrypto aes = new SymmetricCrypto(SymmetricAlgorithm.AES, key);  
  
 long aesStartTime = System.*currentTimeMillis*();  
 encryContent = aes.encryptHex(decryContent);  
 long esaEndTimeEncry = System.*currentTimeMillis*();  
  
 System.*out*.println("aes encrypt...:" + encryContent);  
 System.*out*.println("aes File size after encryption:" + encryContent.length() /1024/1024 + "Mb");  
 System.*out*.println("aes Encryption Time cost:" + (esaEndTimeEncry - aesStartTime) + "ms");  
  
  
 long aesEncryStartTime = System.*currentTimeMillis*();  
 decryContent = aes.decryptStr(encryContent, CharsetUtil.CHARSET\_UTF\_8);  
 long aesEncryEndTime = System.*currentTimeMillis*();  
 System.*out*.println("aes after decryption:" + decryContent);  
 System.*out*.println("aesFile size after decryption:" + decryContent.length() /1024/1024 + "Mb");  
 System.*out*.println("aes decryption time cost:" + (aesEncryEndTime - aesEncryStartTime) + "ms");  
  
  
 // des  
 key = SecureUtil.generateKey(SymmetricAlgorithm.DESede.getValue()).getEncoded();  
 SymmetricCrypto des = new SymmetricCrypto(SymmetricAlgorithm.DESede, key);  
 long desStartTime = System.*currentTimeMillis*();  
 encryContent = des.encryptHex(decryContent);  
 long desEndTimeEncry = System.*currentTimeMillis*();  
  
 System.*out*.println("des encryption..." + encryContent);  
 System.*out*.println("des File size after encryption:" + encryContent.length() /1024/1024 + "Mb");  
 System.*out*.println("des Encryption Time cost:" + (desEndTimeEncry - desStartTime) + "ms");  
  
  
 long desEncryStartTime = System.*currentTimeMillis*();  
 decryContent = des.decryptStr(encryContent, CharsetUtil.CHARSET\_UTF\_8);  
 long desEncryEndTime = System.*currentTimeMillis*();  
 System.*out*.println("des after decryption:" + decryContent);  
 System.*out*.println("des File size after decryption:" + decryContent.length() /1024/1024 + "Mb");  
 System.*out*.println("des decryption time cost:" + (desEncryEndTime - desEncryStartTime) + "ms");\*/  
  
 }  
  
  
  
}

# Результат

## AES-128





Используем функцию, реализованную в соответствии с алгоритмом AES, для шифрования входного открытого текста " 0x32,0x88,0x31,0xe0,0x43,0x5a,0x31,0x37, 0xf6,0x30,0x98,0x07, 0xa8,0x8d,0xa2,0x34", чтобы получить зашифрованный текст. Затем расшифруем зашифрованный текст. Мы обнаружили, что расшифрованный текст совпадает с исходным текстом.

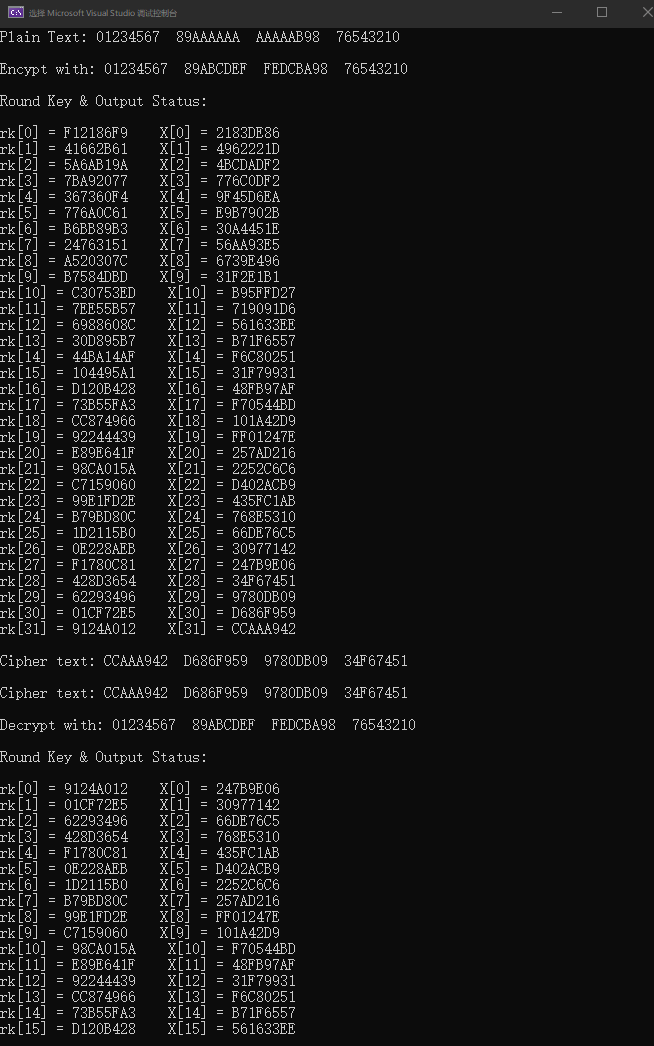
Даже реалтзована программа для зашифровании и расшифравании с помощью AES-128.

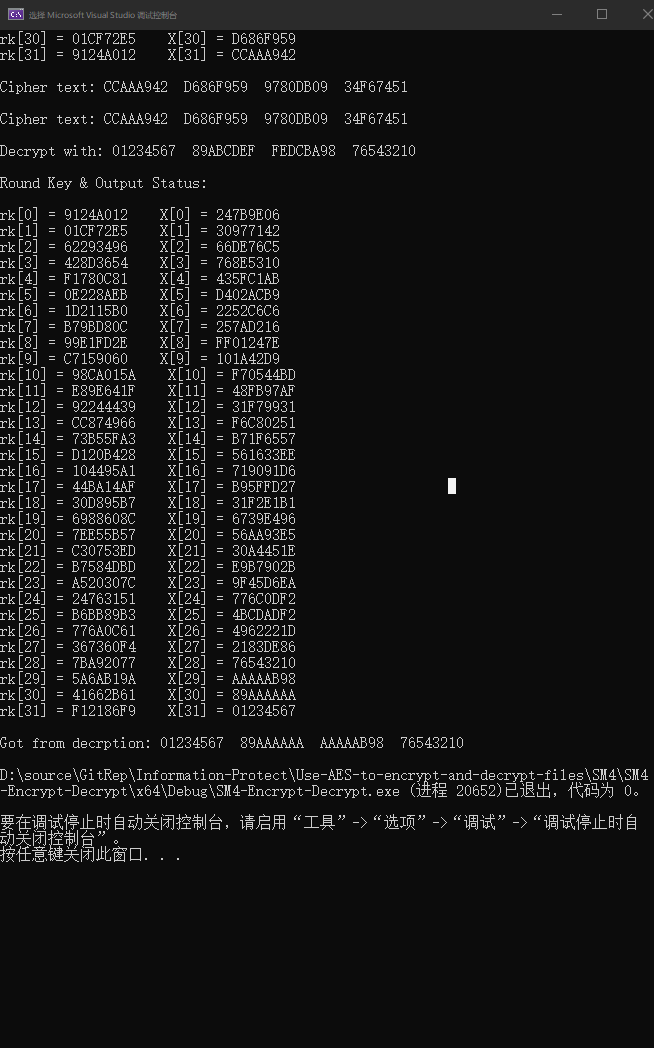
Для тестировании взала 10 фалов с разних размеров. См. Ниже в таблице

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Original File Size (M) | File size after encryption (M) | Encryption Time cost (s) | Decryption Time Cost (s) |
| 1 | 1.639 | 3.335 | 0 | 0 |
| 2 | 3.278 | 6.669 | 0 | 1 |
| 3 | 6.556 | 13.338 | 1 | 2 |
| 4 | 13.111 | 26.676 | 3 | 5 |
| 5 | 26.226 | 53.351 | 6 | 11 |
| 6 | 52.441 | 106.701 | 12 | 23 |
| 7 | 104.882 | 213.401 | 23 | 44.62 |
| 8 | 209.763 | 426.801 | 47 | 95.41 |
| 9 | 839.051 | 1707.204 | 203 | 379.61 |
| 10 | 1048.813 | 2134.005 | 255 | 433.5 |

Заметим, что с увеличением размера файла наблюдается рост временных затрат на проведение указанных операций. Размер зашифрованного файла примерно вдвое больше по сравнению с исходным. В связи с этим, время дешифрования превышает время шифрования, которое также составляет приблизительно двойное значение.

## SM4





Сревнение

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Key Length (bit) | Block Length | Number of Round | Algotirhm Structure |
| AES | 128/192/256 | 128 | 10 | Substitution-Permutation |
| SM4 | 128 | 128 | 32 | Non-balanced Feistel |
| DES | 64 | 64 | 16 | Feistel |

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Original file size | Encryption Time cost (ms) | Decryption Time Cost (ms) |
| SM4 | 5M | 1596 | 841 |
| AES | 5M | 201 | 447 |
| DES | 5M | 317 | 669 |
| SM4 | 620KB | 689 | 172 |
| AES | 620KB | 109 | 186 |
| DES | 620KB | 53 | 132 |
| SM4 | 36KB | 487 | 34 |
| AES | 36KB | 20 | 57 |
| DES | 36KB | 9 | 14 |
| SM4 | 6Bytes | 505 | 0 |
| AES | 6Bytes | 13 | 24 |
| DES | 6Bytes | 1 | 0 |

# Вывод

Наибольшее влияние на производительность DES оказывает функция Фейстеля. Наибольшее влияние на производительность AES оказывает путаница в столбцах. Для SM4 наибольшее влияние на производительность оказывает синтетическая замена.

AES-128 и SM4 имеют некоторые сходства в конструкции.

В соответствии с приведенными выше экспериментальными результатами можно обнаружить, что в контролируемом эксперименте AES-128 значительно превосходит SM4. AES быстрее, чем SM4 и DES, с точки зрения шифрования и дешифрования. Кроме того, AES быстрее шифрует и расшифровывает небольшие данные при работе с небольшими файлами.

Напротив, SM4 и DES значительно медленнее обрабатывают большие файлы, но лучше, чем AES, с точки зрения безопасности. SM4 — это китайский национальный стандарт коммерческого алгоритма шифрования, который, по результатам многочисленных обзоров экспертов, считается высоконадежным, подходящим для использования в различных сценариях и обладающим хорошей масштабируемостью.

Хотя DES больше не рекомендуется, он все еще используется в некоторых старых системах, потому что, хотя безопасность DES не так хороша, как SM4 и AES, в некоторых особых случаях, таких как встроенные устройства, он может быть более подходящим облегченным алгоритмом шифрования.

# Список лителатуры

1. Grabbe J O. The DES algorithm illustrated, [Статья в журналах]. 2010.
2. Seung-Jo Han, Heang-Soo Oh and Jongan Park, "The improved data encryption standard (DES) algorithm," Proceedings of ISSSTA'95 International Symposium on Spread Spectrum Techniques and Applications, Mainz, Germany, [Статья в журналах], 1996, pp. 1310-1314 vol.3, doi: 10.1109/ISSSTA.1996.563518.
3. Wu Wenling, Feng Dengguo, «Brief Commentary on 21st Century European Data Encryption Standard Candidate Algorithms», [Статья в журналах]. 2001, 12 (1): 49-55
4. Raghavan N.S, «AES: Cryptography advances into the future», [Статья в журналах]., Java World, 2000, 12(4): 47-51.
5. « Federal Information Processing Standards Publication 197 November 26, 2001 Specification for the ADVANCED ENCRYPTION STANDARD (AES)», [Монография]
6. «What Is AES Advanced Encryption Standard? », [Электроный ресурс], URL: <https://www.wallarm.com/what/what-is-aes-advanced-encryption-standard>, Время обращения: 27.02.2023
7. <http://www.codeofchina.com/standard/GMT0002-2012.html> [Архивная копия](https://web.archive.org/web/20160304105303/http:/www.codeofchina.com/standard/GMT0002-2012.html) от 4 марта 2016 на [Wayback Machine](https://ru.wikipedia.org/wiki/Wayback_Machine" \o "Wayback Machine) GM/T 0002-2012 SM4 Block Cipher Algorithm
8. IEEE Standard for information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements-Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements, <https://standards.ieee.org/ieee/802.11i/3127/>