Расширенный стандарт шифрования (AES)

AES представляет собой представляет собой алгоритм шифрования 128-битных блоков данных ключами по 128, 192 и 256 бит, принятый в качестве стандарта шифрования правительством США по результатам конкурса AES.

AES реализует архитектуру «квадрат» (все преобразования происходят в рамках одного квадрата) и принципы SPN.

Сеть подстановки-перестановки (SPN), является серией связанных математических операций, когда на вход принимается блок данных с ключом и к ним применяется несколько чередующихся раундов блоков подстановки и блоков перестановки для создания блока шифротекста. Обычно эти преобразования представляют собой операции, которые машина может быстро обработать, такие как (XOR) и побитовый сдвиг.

Особенности шифра AES:

— ориентирован в основном на реализацию с 8-разрядными процессорами;

— все раундовые преобразования выполняются в конечных полях, что допускает простую реализацию на различных платформах;

— высокая скорость и низкие требования к оперативной памяти;

в настоящее время не известно ни одной практической атаки, которая позволила бы кому-либо, не знающему ключа, прочитать данные, зашифрованные AES при правильной реализации.

Этапы реализации алгоритма шифрования (AES)

Терминология (основная)

Байт – последовательность из 8 битов. В контексте данного алгоритма байт рассматривается как элемент поля Галуа. Операции над байтами производятся как над элементами поля Галуа $GF(2^8)$.

Слово – последовательность из четырех байтов.

Блок – последовательность из 16 байтов, над которой оперирует алгоритм. Блок служит входными и выходными данными алгоритма. Байты в блоке нумеруются с нуля.

Ключ – последовательность из 16, 24 или 32 байтов, используемая в качестве ключа шифрования. Байты в ключе нумеруются с нуля. Ключ, наряду с блоком, является входными данными алгоритма.

Порядок байтов в блоке:

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

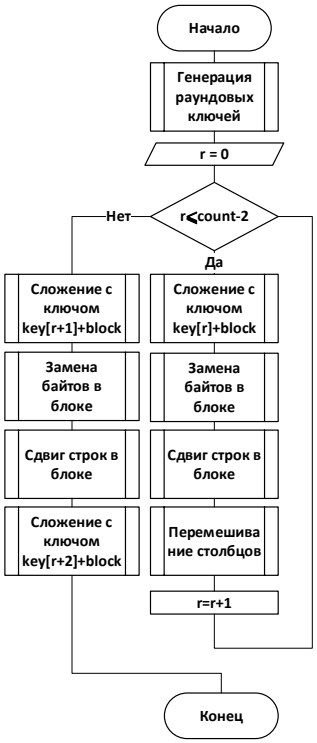
Раунд – цикл(итерация, основной шаг) преобразования над блоком. Количество раундов зависит от длины ключа. (128–10, 192–12, 256–14)

Раундовый ключ – ключ, применяемый в текущем раунде. Вычисляется для каждого раунда.

Таблица подстановок/обратная таблица подстановок – таблица, задающая биективное отображение байта в байт.

Далее будет представлен один из вариантов реализации алгоритма AES для ключа в 128 бит и блока в 16 байт.

Шифрование



Блок-схема шифрования (алгоритм AES)

Зашифрование блока данных. Код на C#:

```
public override void EncodeBlock(Word[] block, IEnumerable<Word[]> currentKey)
{
    var key = currentKey.ToList();
    for (int i = 0; i <= countRound - 2; i++)
    {
        AddRoundKey(key[i].ToList(), block);
        SubBytes(block);
        ShiftRows(block);
        MixColumns(block);
    }
    AddRoundKey(key[countRound - 1].ToList(), block);
    SubBytes(block);
    ShiftRows(block);
    AddRoundKey(key[countRound].ToList(), block);
}
```

Терминология

numBWord – количество байт в слове.

countRound – количество раундов.

sizeBlock – количество слов в одном блоке данных.

sizeKey – количество слов в раундовом ключе.

Функция генерации раундовых ключей (ExpandKey)

Процедура ExpandKey создает countRound+1 раундовых ключей.

Описание действий	Фрагменты кода на C#

Сформировать из начального ключа массив слов.	`for (int i = 0; i < startkey.Count; i += 4)`
	`{`
	`  newkey.Add(new Word(key, i));`
	`}`

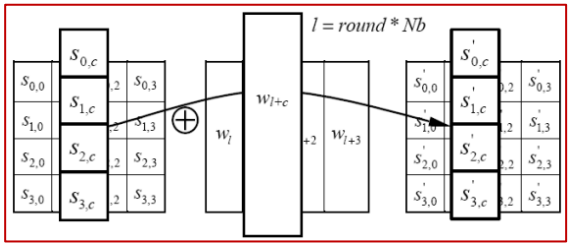
Провести ряд операций над массивом слов. Здесь используются следующие функции:	`for (int i = sizeKey; i < countWordInKey; i++)`
	`{`
	`  Word key = newkey[i - 1];`
	`  if (i % sizeKey == 0)`
	`  {`
	`  key = key.RotWordLeft();`

RotWordLeft – сдвиг байт в слове на 1 влево.
SubWord – замена байт в слове по таблице замен.
Pow – возведение в степень.
DivOfPolyGF – деление в поле GF(2^8).
XOR – побитовое сложение.
	key = key.SubWord();
	int byte1 = (int)System.Math.Pow(2, i / sizeKey);
	if (byte1 > 128)
	{
	byte1 = MathematicsGF.DivOfPolyGF(byte1, 283)[1];
	}
	key = key.XOR(new Word(new(Byte1: byte1)));
	}
	else if ((sizeKey > 6) && (i % sizeKey == 4))
	{
	key = key.SubWord();
	}
	key = newkey[i - sizeKey].XOR(key);
	newkey.Add(key);
	}

Формирование блоков раундовых ключей.
Здесь используются следующие функции:
XorBlocks – сложение блоков слов.
	for (int i = 0, x = 0; i < numByteOfBlock; i++)
	{
	result.Add(new Word[sizeBlock]);
	for (int j = 0; j < sizeBlock; j++, x++)
	{
	result[i][j] = newkey[x];
	}
	}
	for (int i = countRound + 1; i < result.Count - 1; i++)
	{
	result[i] = Word.XorBlocks(result[i], result[i + 1]);
	}
	RoundKeys = (result.ToArray())[0..(countRound + 1)];

Функция сложения с раундовым ключом (AddRounKey)

В преобразовании AddRounKey слова раундового ключа прибавляются к словам блоков входных данных с помощью побитовой операции XOR.



Над каждым столбцом операция проводится отдельно.

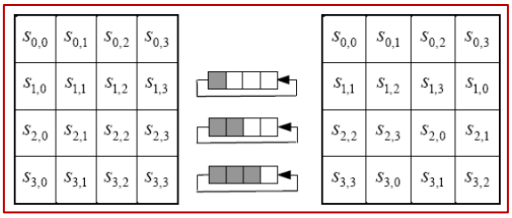
Пример кода на C#:

```
void AddRoundKey(List<Word> currentKey, Word[] block)
{
    for (int i = 0; i < block.Length; i++)
    {
        block[i] = block[i].XOR(currentKey[i]);
    }
}
```

```
public Word XOR(Word word2)
{
    return new Word
    {
        Byte1 = Byte1 ^ word2.Byte1,
        Byte2 = Byte2 ^ word2.Byte2,
        Byte3 = Byte3 ^ word2.Byte3,
        Byte4 = Byte4 ^ word2.Byte4
    };
}
```

Функция преобразования (ShiftRows)

Данное преобразование заключается в циклическом сдвиге строк блока входных данных.



Функция преобразования (MixColumns)

Преобразование MixColumns заключается в умножении квадратной матрицы 4-го порядка на каждый столбец блока входных данных. Умножение производится в поле Галуа(2^8). Над каждым столбцом операция производится отдельно.

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix}$$

Пример кода на C#:

```
private static void MixColumns(Word[] block)
{
    int[,] matrix = new int[,]
    {
              { 0x02, 0x03, 0x01, 0x01 },
              { 0x01, 0x02, 0x03, 0x01 },
              { 0x01, 0x01, 0x02, 0x03 },
              { 0x03, 0x01, 0x01, 0x02 }
    };
    foreach (Word word in block)
    {
        int[,] matrixWord = new int[,]
        {
              {word.Byte1 },
              {word.Byte2 },
              {word.Byte3 },
              {word.Byte4 }
        };

        matrixWord = MathematicsGF.MultOfMatrixGF(matrix, matrixWord, 283);

        word.Byte1 = (byte)matrixWord[0, 0];
        word.Byte2 = (byte)matrixWord[1, 0];
        word.Byte3 = (byte)matrixWord[2, 0];
        word.Byte4 = (byte)matrixWord[3, 0];
    }
}
```

Здесь используется функция MultOfMatrixGF – умножение матриц в поле GF(2^n).

Функция преобразования (SubBytes)

Преобразование SubBytes заключается в замене каждого байта[x, y] блока данных на другой в соответствии с таблицей подстановок.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Генерация таблицы подстановок. Блок-схема:

Генерация таблицы подстановок. Код на C#:

```
public static byte SubBytes(byte myByte)
{
    int[,] matrix = new int[,]
    {
                {1, 0, 0, 0, 1, 1, 1, 1 },
                {1, 1, 0, 0, 0, 1, 1, 1 },
                {1, 1, 1, 0, 0, 0, 1, 1 },
                {1, 1, 1, 1, 0, 0, 0, 1 },
                {1, 1, 1, 1, 1, 0, 0, 0 },
                {0, 1, 1, 1, 1, 1, 0, 0 },
                {0, 0, 1, 1, 1, 1, 1, 0 },
                {0, 0, 0, 1, 1, 1, 1, 1 }
    };
    int b = MathematicsGF.InversionPolynomialGF(myByte, 283);
    int[,] c = MathematicsGF.MultOfMatrixGF(matrix, MathematicsGF.CreatBITMatrixFromPoly(b, 8), 7);
    int[,] d = MathematicsGF.XorMatrix(c, MathematicsGF.CreatBITMatrixFromPoly(0x63, 8));
    byte result = (byte)MathematicsGF.CreatPolyFromBITMatrix(d);
    return result;
}
```

Здесь используются функции:

InversionPolynomialGF – нахождение инверсного сомножителя.

MultOfMatrixGF – умножение матриц в поле GF(2^8).