Блочный криптографический алгоритм (МАГМА – ГОСТ 32.12-2018)

Алгоритм Магма применяет сеть Фейстеля для шифрования данных. **Сеть Фейстеля** состоит из ячеек, на вход каждой ячейки поступают данные и ключевая информация. После преобразования каждой ячейки получаются изменённые данные и изменённый ключ. Все ячейки сети однотипны и вместе составляют некую многократно повторяющуюся структуру. Ключ меняется в каждом раунде. При шифровании и расшифровании выполняются одни и те же операции (отличие только в порядке ключей).

ГОСТ 34.12-2018 оперирует **64**-битными блоками и ключом длиной **256** бит. Ключ разбивается на **8** подключей, каждый из которых используется в собственном раунде сети. После нескольких раундов получается блок преобразованных данных.

Достоинствами сети Фейстеля можно считать простоту аппаратной и программной реализации, а недостатком – то, что за один раунд шифруется лишь половина исходного блока.

Особенности стандарта ГОСТ 34.12-2018 (Магма):

– значительная скорость работы на современных машинах (но, как и любой шифр на основе сети Фейстеля, он будет медленнее шифров, реализующих SP-сеть);

– одинаковый цикл преобразования и защита от ложных данных (выработка имитовставки) и во всех алгоритмах стандарта.

Этапы реализации алгоритма шифрования (МАГМА – ГОСТ 32.12-2018)

Терминология (основная)

Байт – последовательность из 8 битов. В контексте данного алгоритма байт рассматривается как элемент поля Галуа.

Слово – последовательность из 4 байтов.

Блок – последовательность из 2-х слов (8-байтов, 64-бита), над которой оперирует алгоритм. Блок служит входными и выходными данными алгоритма. Байты в блоке нумеруются с нуля.

Ключ – последовательность из 1 слова (4-байта, 32-бита), используемая в качестве ключа шифрования. Байты в ключе нумеруются с нуля. Ключ, наряду с блоком, является входными данными алгоритма.

Порядок байтов в блоке:

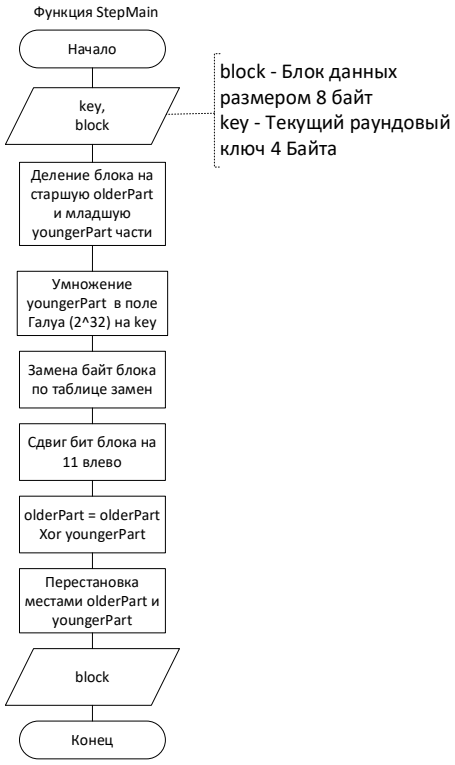
0	4
1	5
2	6
3	7

Раунд – цикл(итерация, основной шаг) преобразования над блоком. Количество раундов – 32.

Раундовый ключ – 8 ключей, применяемых в текущем раунде. Вычисляются для каждого раунда.

Таблица замен – является матрица 8×16, содержащая 4-битовые элементы, которые можно представить в виде целых чисел от 0 до 15. Строки таблицы замен называются узлами замен, они должны содержать различные значения, то есть каждый узел замен должен содержать 16 различных чисел от 0 до 15 в произвольном порядке. Общий объем таблицы замен – 8 * 16 * 4 = 512-бит, 64 байта.

Этапы реализации алгоритма шифрования – ОСНОВНОЙ ШАГ



Блок-схема основного шага криптопреобразования (шифрование).

Функция StepMain.

Основной шаг криптопреобразования по своей сути является оператором, определяющим преобразование 64-битового блока данных. Дополнительным параметром этого оператора является 32-битовый блок, в качестве которого используется какой-либо элемент ключа.

Зашифрование блока данных. Код на C#:

```
public override void EncodeBlock(Word[] block, IEnumerable<Word[]> currentKey)
{
    var key = currentKey.ToList();
    void Step(Word key)
    {
        Word olderPart = block[0];
        Word youngerPart = block[1];
        youngerPart = youngerPart.MultGF(key, 4299161607);
        #region SubBytes
        #region RotLeft() << 11
        olderPart = youngerPart.XOR(olderPart);
        block[0] = youngerPart;
        block[1] = olderPart;
    }

    for (int i = 0; i < countRound; i++)
    {
        if (i != 3)
        {
            for (int j = 0; j < countKeys; j++)
            {
                Step(key[j][0]);
            }
        }
        else
        {
            for (int j = countKeys - 1; j >= 0; j--)
            {
                Step(key[j][0]);
            }
        }
    }
    currentKey = key;
}
```

Терминология

currentKey – текущий ключ раунда.

olderPart – старшая половина блока.

youngerPart – младшая половина блока.

countRound – кол-во раундов шифрования (4).

countKeys – кол-во раундовых ключей (8).

Основные функции – функция SubBytes

Описание действия	Фрагмент кода на C#

Фрагмент SubBytes – замена байтов в младшей половины блока в соответствии с алгоритмом, имитирующим таблицу подстановок.	`string stringView = youngerPart.GetStringView16();`
	`string arrayS = string.Empty;`
	`for (int i = 0; i < stringView.Length; i++)`
	`{`
	`  byte res = SubBit(Convert.ToByte(stringView[i].ToString(), 16), i);`
	`  arrayS += Convert.ToString(res, 16);`
	`}`
	`youngerPart = new Word`
	`{`
	`  Byte1 = Convert.ToByte(arrayS[0..2], 16),`
	`  Byte2 = Convert.ToByte(arrayS[2..4], 16),`
	`  Byte3 = Convert.ToByte(arrayS[4..6], 16),`
	`  Byte4 = Convert.ToByte(arrayS[6..8], 16)`
	`};`

Функция GetStringView16 – шестнадцатеричное представление блока.	`public string GetStringView16()`
	`{`
	`  static string Step(int myByte)`
	`  {`
	`  string resultByte = Convert.ToString(myByte, 16);`
	`  while (resultByte.Length < 2) { resultByte = "0" + resultByte;`
	`}`
	`  return resultByte;`
	`  }`
	`  string result = Step(Byte1) + Step(Byte2) + Step(Byte3) +`
	`Step(Byte4);`
	`  return result;`
	`}`

Функция SubBit – замена битов байта.	`private static byte SubBit(byte myByte, int index)`
	`{`
	`  string resultBit = Convert.ToString(myByte, 2);`
	`  while (resultBit.Length < 4) { resultBit = '0' + resultBit; }`
	`  for (int i = 0; i <= index; i++)`
	`  {`
	`  resultBit = GOST89.RotLeft(resultBit.ToArray());`
	`  }`
	`  byte res = Convert.ToByte(resultBit, 2);`
	`  res ^= (byte)(index + 1);`
	`  return res;`
	`}`

Функция RotLeft – сдвиг влево на 1.	`private static string RotLeft(char[] myStr)`
	`{`
	`  char cup = myStr[0];`
	`  string result = string.Empty;`
	`  for (int i = 1; i < myStr.Length; i++)`
	`  {`
	`  result += myStr[i];`
	`  }`
	`  result += cup;`
	
	`  return result;`
	`}`

Фрагмент SubBytes. Алгоритм:

1. Получить шестнадцатеричное представление младшей половины блока (**GetStringView16**).
2. Заменить каждый бит в соответствии с алгоритмом (**SubBit**).
3. Получить из шестнадцатеричного представления (строки) блок «слов (Word)».

Основные функции – функция RotLeft() << 11

Описание действия	Фрагмент кода на C#

Фрагмент RotLeft() << 11 – сдвиг младшей половины блока на 11 бит влево.	`stringView = youngerPart.GetStringView2();`
	`for (int i = 0; i < 11; i++)`
	`{`
	`  stringView = RotLeft(stringView.ToArray());`
	`}`
	`stringView = Convert.ToString((long)Convert.ToUInt64(stringView, 2), 16);`
	`while (stringView.Length < 8)`
	`{`
	`  stringView = Convert.ToString(0) + stringView;`
	`}`
	`youngerPart = new Word`
	`{`
	`  Byte1 = Convert.ToByte(stringView[0..2], 16),`
	`  Byte2 = Convert.ToByte(stringView[2..4], 16),`
	`  Byte3 = Convert.ToByte(stringView[4..6], 16),`
	`  Byte4 = Convert.ToByte(stringView[6..8], 16)`
	`};`

Функция GetStringView2 – двоичное представление блока данных.	`public string GetStringView2()`
	`{`
	`  static string Step(int myByte)`
	`  {`
	`  string resultByte = Convert.ToString(myByte, 2);`
	`  while (resultByte.Length < 8) { resultByte = "0" + resultByte;`
	`}`
	`  return resultByte;`
	`  }`
	`  string result = Step(Byte1) + Step(Byte2) + Step(Byte3) + Step(Byte4);`
	`  return result;`
	`}`

Фрагмент RotLeft() << 11. Алгоритм:

1. Получить двоичное представление младшей половины блока (**GetStringView2**).
2. Сдвинуть двоичное представление младшей половины блока на 11 бит влево.
3. Получить из двоичного представления (строки) блок «слов (Word)».

Основные функции – функция StepMain

Код на C#:

```
void Step(Word key)
{
    Word olderPart = block[0];
    Word youngerPart = block[1];
    youngerPart = youngerPart.MultGF(key, 4299161607);
    #region SubBytes
    #region RotLeft() << 11
    olderPart = youngerPart.XOR(olderPart);
    block[0] = youngerPart;
    block[1] = olderPart;
}
```

Алгоритм:

1. Разделить входной блок на старшую и младшую части.
2. Умножить младшую часть блока на ключ (в поле $GF(2^{32})$) по модулю 4299161607_{10}.
3. Применить преобразование SubBytes.
4. Применить преобразование RotLeft() << 1.
5. Сложить старшую и младшую части.
6. Поменять части местами.

Блок-схема блочного криптоалгоритма Магма