Содержание

ВВЕДЕНИЕ

Актуальность темы: актуальность темы "Программная реализация алгоритма шифрования IDEA" обусловлена тем, что надежная передача данных является одной из самых важных задач на сегодняшний день, а алгоритм, реализацией которого я занималась, используется в таких областях как:

- шифрование аудио и видео данных для кабельного телевидения, видеоконференций, дистанционного обучения, VoIP.
- защита коммерческой и финансовой информации, отражающей конъюнктурные колебания
- линии связи через модем, роутер или ATM линию, GSM технологию
- смарт-карты
- общедоступный пакет конфиденциальной версии электронной почты PGP v2.0 и (опционально) в OpenPGP

Объект исследования: алгоритм шифрования IDEA.

Предмет исследования: математическое и программное представления международного алгоритма шифрования данных

Цель работы: изучить симметричный блочный алгоритм шифрования данных, IDEA, его работу в режиме обратной связи по шифротексту, CFB, и программно реализовать его на одном из языков программирования высокого уровня.

Метод исследования: для достижения поставленной цели в работе применялись теоритичексие (анализ и синтез), эмпирические (тестирование) и математические (программирование и визуализация) методы исследования.

1. ТЕОРИТИЧЕСКАЯ ЧАСТЬ

1.1 История.

IDEA (англ. International Data Encryption Algorithm, международный алгоритм шифрования данных) — симметричный блочный алгоритм шифрования данных. Первую версию алгоритма разработали в 1990 году Лай Сюэцзя и Джеймс Мэсси из Швейцарского института ETH Zürich в качестве замены DES (англ. Data Encryption Standard, стандарт шифрования данных) и назвали её PES (англ. Proposed Encryption Standard, предложенный стандарт шифрования). Затем алгоритм был улучшен с целью усиления криптостойкости и назван IPES (англ. Improved Proposed Encryption Standard, улучшенный предложенный стандарт шифрования). Через год его переименовали в IDEA.

1.2 Описание симметричного алгоритма IDEA.

Так как IDEA использует 128-битный ключ и 64-битный размер блока, открытый текст разбивается на блоки по 64 бит. Если такое разбиение невозможно, последний блок дополняется различными способами определённой последовательностью бит. Для избежания утечки информации о каждом отдельном блоке используются различные режимы шифрования. Каждый исходный незашифрованный 64-битный блок делится на четыре подблока по 16 бит каждый, так как все алгебраические операции, использующиеся в процессе шифрования, совершаются над 16-битными числами. Для шифрования и расшифрования IDEA использует один и тот же алгоритм.

Фундаментальным нововведением в алгоритме является использование операций из разных алгебраических групп, а именно:

- сложение по модулю 2^{16}
- умножение по модулю $2^{16} + 1$
- побитовое исключающее ИЛИ (XOR)

Применение этих трех операций затрудняет криптоанализ IDEA по сравнению с DES, который основан исключительно на операции исключающее ИЛИ,

а также позволяет отказаться от использования S-блоков и таблиц замены. IDEA является модификацией сети Фейстеля.

1.3 Генерация ключей.

Из 128-битного ключа для каждого из восьми раундов шифрования генерируется по шесть 16-битных подключей, а для выходного преобразования генерируется четыре 16-битных подключа. Всего потребуется 52 = 8 x 6 + 4 различных подключей по 16 бит каждый. Процесс генерации пятидесяти двух 16-битных ключей заключается в следующем:

- Первым делом, 128-битный ключ разбивается на восемь 16-битных блоков. Это будут первые восемь подключей по 16 бит каждый — $(K_1^{(1)}, K_2^{(1)}, K_3^{(1)}, K_4^{(1)}, K_5^{(1)}, K_6^{(1)}, K_1^{(2)}, K_2^{(2)})$
- Затем этот 128-битный ключ циклически сдвигается влево на 25 позиций, после чего новый 128-битный блок снова разбивается на восемь 16-битных блоков. Это уже следующие восемь подключей по 16 бит каждый — $(K_3^{(2)}, K_4^{(2)}, K_5^{(2)}, K_6^{(2)}, K_1^{(3)}, K_2^{(3)}, K_3^{(3)}, K_4^{(3)})$
- Процедура циклического сдвига и разбивки на блоки продолжается до тех пор, пока не будут сгенерированы все 52 16-битных подключа.

Таблица подключей для каждого раунда	
Номер раунда	Подключи
1	$K_1^{(1)}, K_2^{(1)}, K_3^{(1)}, K_4^{(1)}, K_5^{(1)}, K_6^{(1)}$
2	$K_1^{(2)}, K_2^{(2)}, K_3^{(2)}, K_4^{(2)}, K_5^{(2)}, K_6^{(2)}$
3	$K_1^{(3)}, K_2^{(3)}, K_3^{(3)}, K_4^{(3)}, K_5^{(3)}, K_6^{(3)}$
4	$K_1^{(4)}, K_2^{(4)}, K_3^{(4)}, K_4^{(4)}, K_5^{(4)}, K_6^{(4)}$
5	$K_1^{(5)}, K_2^{(5)}, K_3^{(5)}, K_4^{(5)}, K_5^{(5)}, K_6^{(5)}$
6	$K_1^{(6)}, K_2^{(6)}, K_3^{(6)}, K_4^{(6)}, K_5^{(6)}, K_6^{(6)}$
7	$K_1^{(7)}, K_2^{(7)}, K_3^{(7)}, K_4^{(7)}, K_5^{(7)}, K_6^{(7)}$
8	$K_1^{(8)}, K_2^{(8)}, K_3^{(8)}, K_4^{(8)}, K_5^{(8)}, K_6^{(8)}$
выходное преобразование	$K_1^{(9)}, K_2^{(9)}, K_3^{(9)}, K_4^{(9)}$

1.4 Шифрование.

Структура алгоритма IDEA показана на рисунке. Процесс шифрования состоит из восьми одинаковых раундов шифрования и одного выходного преобразования.

Исходный незашифрованный текст делится на блоки по 64 бита. Каждый такой блок делится на четыре подблока по 16 бит каждый. На рисунке эти подблоки обозначены D_1, D_2, D_3, D_4.

В каждом раунде используются свои подключи согласно таблице подключей. Над 16-битными подключами и подблоками незашифрованного текста производятся следующие операции:

- умножение по модулю $2^{16} + 1 = 65537$
- сложение по модулю 2^{16}
- побитовое исключающее ИЛИ

В конце каждого раунда шифрования имеется четыре 16-битных подблока, которые затем используются как входные подблоки для следующего раунда шифрования.

Выходное преобразование представляет собой укороченный раунд, а именно, четыре 16-битных подблока на выходе восьмого раунда и четыре соответствующих подключа подвергаются операциям:

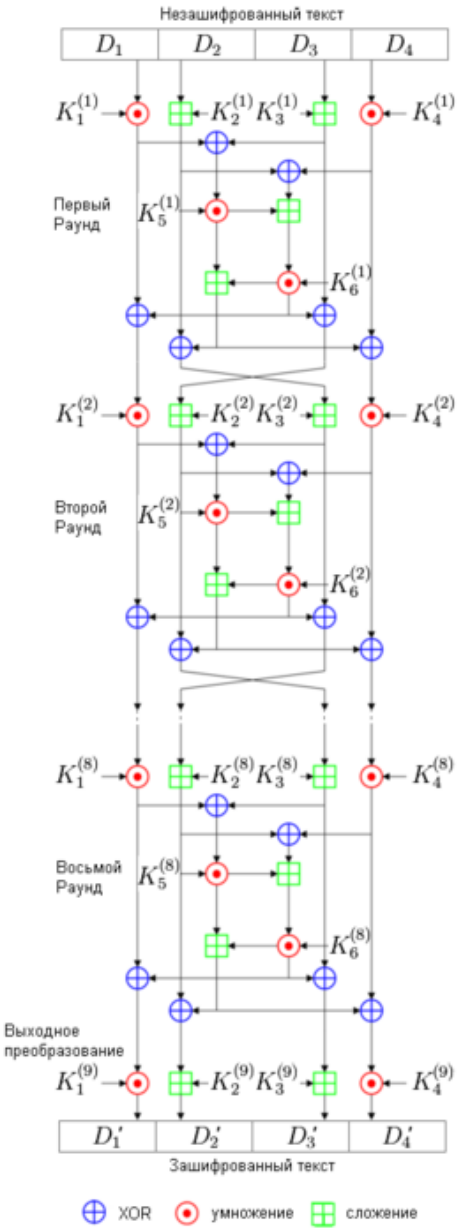

Рис. 1: Схема шифрования IDEA.

- умножение по модулю $2^{16}+1$
- сложение по модулю 2^{16}

После выполнения выходного преобразования конкатенация подблоков $D_1^{'}, D_2^{'}, D_3^{'}$ и $D_4^{'}$ представляет собой зашифрованный текст. Затем берется следующий 64-битный блок незашифрованного текста и алгоритм шифрования повторяется. Так продолжается до тех пор, пока не зашифруются все 64-битные блоки исходного текста.

1.5 Математическое описание.

- Блок открытого текста размером 64 бит делится на четыре равных подблока размером по 16 бит (D_1^0, D_2^0, D_3^0 и D_4^0)
- Для каждого раунда ($i = 1 \dots 8$) вычисляются:

$$
\begin{aligned}
A^{(i)} &= D_1^{(i-1)} * K_1^{(i)} \\
B^{(i)} &= D_2^{(i-1)} * K_2^{(i)} \\
C^{(i)} &= D_3^{(i-1)} * K_3^{(i)} \\
D^{(i)} &= D_4^{(i-1)} * K_4^{(i)} \\
E^{(i)} &= A^{(i)} \oplus C^{(i)} \\
F^{(i)} &= B^{(i)} \oplus D^{(i)}
\end{aligned}
$$

$$
\begin{aligned}
D_1^{(i)} &= A^{(i)} \oplus ((F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)}) \\
D_2^{(i)} &= C^{(i)} \oplus ((F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)}) \\
D_3^{(i)} &= B^{(i)} \oplus (E^{(i)} * K_5^{(i)} + (F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)})) \\
D_4^{(i)} &= D^{(i)} \oplus (E^{(i)} * K_5^{(i)} + (F^{(i)} + E^{(i)} * K_5^{(i)}) * K_6^{(i)}))
\end{aligned}
$$

Результатом выполнения восьми раундов будут следующие четыре подблока $(D_1^{(8)}, D_2^{(8)}, D_3^{(8)}, D_4^{(8)})$

- Выполняется выходное преобразование ($i = 9$):

$$
\begin{aligned}
D_1^{(9)} &= D_1^{(8)} * K_1^{(9)} \\
D_2^{(9)} &= D_3^{(8)} * K_2^{(9)}
\end{aligned}
$$

$D_3^{(9)} = D_2^{(8)} * K_3^{(9)}$
$D_4^{(9)} = D_4^{(8)} * K_4^{(9)}$

Результатом выполнения выходного преобразования является зашифрованный текст $(D_1^{(9)}, D_2^{(9)}, D_3^{(9)}, D_4^{(9)})$

1.6 Расшифровка.

Метод вычисления, использующийся для расшифровки текста по существу такой же, как и при его шифровании. Единственное отличие состоит в том, что для расшифровки используются другие подключи. В процессе расшифровки подключи должны использоваться в обратном порядке. Первый и четвёртый подключи i-го раунда расшифровки получаются из первого и четвёртого подключа (10-i)-го раунда шифрования мультипликативной инверсией. Для 1-го и 9-го раундов второй и третий подключи расшифровки получаются из второго и третьего подключей 9-го и 1-го раундов шифрования аддитивной инверсией. Для раундов со 2-го по 8-й второй и третий подключи расшифровки получаются из третьего и второго подключей с 8-го по 2-й раундов шифрования аддитивной инверсией. Последние два подключа i-го раунда расшифровки равны последним двум подключам (9-i)-го раунда шифрования. Мультипликативная инверсия подключа K обозначается 1/K и $(1/K) * K = 1 \ mod \ (2^{16} + 1)$. Так как $2^{16} + 1$ — простое число, каждое целое не равное нулю K имеет уникальную мультипликативную инверсию по модулю $2^{16} + 1$. Аддитивная инверсия подключа K обозначается -K и $-K + K = 0 \ mod \ (2^{16})$.

Таблица подключей для каждого раунда	
Номер раунда	Подключи
1	$1/K_1^{(9)}, -K_2^{(9)}, -K_3^{(9)}, 1/K_4^{(9)}, K_5^{(8)}, K_6^{(8)}$
2	$1/K_1^{(8)}, -K_3^{(8)}, -K_2^{(8)}, 1/K_4^{(8)}, K_5^{(7)}, K_6^{(7)}$
3	$1/K_1^{(7)}, -K_3^{(7)}, -K_2^{(7)}, 1/K_4^{(7)}, K_5^{(6)}, K_6^{(6)}$
4	$1/K_1^{(6)}, -K_3^{(6)}, -K_2^{(6)}, 1/K_4^{(6)}, K_5^{(5)}, K_6^{(5)}$
5	$1/K_1^{(5)}, -K_3^{(5)}, -K_2^{(5)}, 1/K_4^{(5)}, K_5^{(4)}, K_6^{(4)}$
6	$1/K_1^{(4)}, -K_3^{(4)}, -K_2^{(4)}, 1/K_4^{(4)}, K_5^{(3)}, K_6^{(3)}$
7	$1/K_1^{(3)}, -K_3^{(3)}, -K_2^{(3)}, 1/K_4^{(3)}, K_5^{(2)}, K_6^{(2)}$
8	$1/K_1^{(2)}, -K_3^{(2)}, -K_2^{(2)}, 1/K_4^{(2)}, K_5^{(1)}, K_6^{(1)}$
выходное преобразование	$1/K_1^{(1)}, -K_2^{(1)}, -K_3^{(1)}, 1/K_4^{(1)}$

1.7 Режимы шифрования.

IDEA является блочным алгоритмом шифрования, работающим с блоками по 64 бита. При несовпадении размера шифруемого текста с этим фиксированным размером, блок дополняется до 64.

Алгоритм используется в одном из следующих режимов шифрования

- Режим электронной кодовой книги (англ. Electronic Codebook (ECB))
- Режим сцепления блоков шифротекста (англ. Cipher Block Chaining (CBC))
- Режим обратной связи по шифротексту (англ. Cipher Feedback (CFB))
- Режим обратной связи по выходу (англ. Output Feedback (OFB))

1.8 Криптостойкость.

В алгоритме IDEA использует 64-битные блоки. Длина блока должна быть достаточной, чтобы скрыть статистические характеристики исходного сообщения. Но с увеличением размера блока экспоненциально возрастает сложность реализации криптографического алгоритма. В алгоритме IDEA используется 128-битный ключ. Длина ключа должна быть достаточно большой, чтобы предотвратить возможность перебора ключа. Для вскрытия 128-битного ключа полным перебором ключей при условии, что известен открытый и соответствующий ему зашифрованный текст, потребуется 2^{128} шифрований. При такой длине ключа IDEA считается довольно безопасным. Высокая криптостойкость IDEA обеспечивается также такими характеристиками:

- запутывание — шифрование зависит от ключа сложным и запутанным образом
- рассеяние — каждый бит незашифрованного текста влияет на каждый бит зашифрованного текста

Лай Сюэцзя и Джеймс Мэсси провели тщательный анализ IDEA с целью выяснения его криптостойкости к дифференциальному криптоанализу. Для этого ими было введено понятие марковского шифра и продемонстрировано, что

устойчивость к дифференциальному криптоанализу может быть промоделирована и оценена количественно. Линейных или алгебраических слабостей у IDEA выявлено не было. Попытка вскрытия с помощью криптоанализа со связанными ключами, проведенная Бихамом, также не увенчалась успехом.

Существуют успешные атаки, применимые к IDEA с меньшим числом раундов (полный IDEA имеет 8.5 раундов). Успешной считается атака, если вскрытие шифра с её помощью требует меньшего количества операций, чем при полном переборе ключей. Метод вскрытия Вилли Майера оказался эффективнее вскрытия полным перебором ключей только для IDEA с 2 раундами. Методом «встреча посередине» был вскрыт IDEA с 4,5 раундами. Для этого требуется знание всех 2^{64} блоков из словаря кодов и сложность анализа составляет 2^{112} операций. Лучшая атака на 2007 год применима ко всем ключам и может взломать IDEA с 6-ю раундами.

1.9 Режим обратной связи по шифротексту

Режим обратной связи по шифротексту, режим гаммирования с обратной связью (англ. Cipher Feedback Mode, CFB) — один из вариантов использования симметричного блочного шифра, при котором для шифрования следующего блока открытого текста он складывается по модулю 2 с перешифрованным (блочным шифром) результатом шифрования предыдущего блока.

Шифрование может быть описано следующим образом:

$C_0 = IV$
$C_i = E_k(C_{i-1}) \oplus P_i$
$P_i = E_k(C_{i-1}) \oplus C_i$

где i — номера блоков, IV — вектор инициализации (синхропосылка), C_i и P_i — блоки зашифрованного и открытого текстов соответственно, а E_k — функция блочного шифрования.

Вектор инициализации IV, как и в режиме сцепления блоков шифротекста, можно делать известным, однако он должен быть уникальным.

Ошибка, которая возникает в шифротексте при передаче (например, из-за помех), сделает невозможным расшифровку как блока, в котором ошибка произошла, так и следующего за ним, однако не распространяется на после-

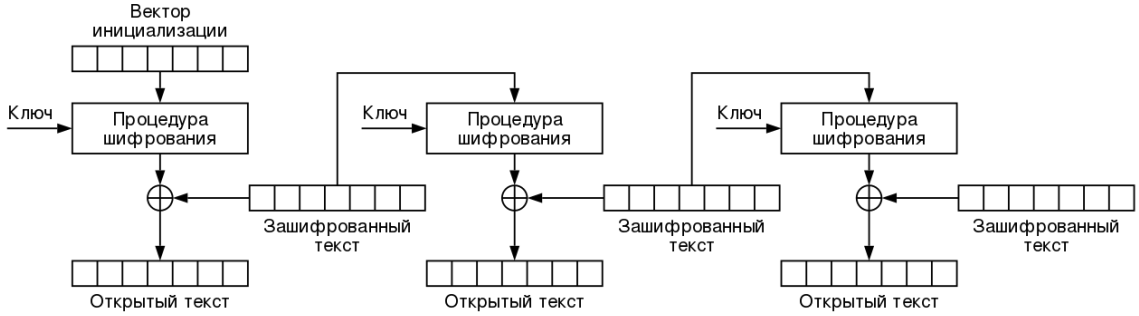

Рис. 2: Шифрование в режиме обратной связи по шифротексту

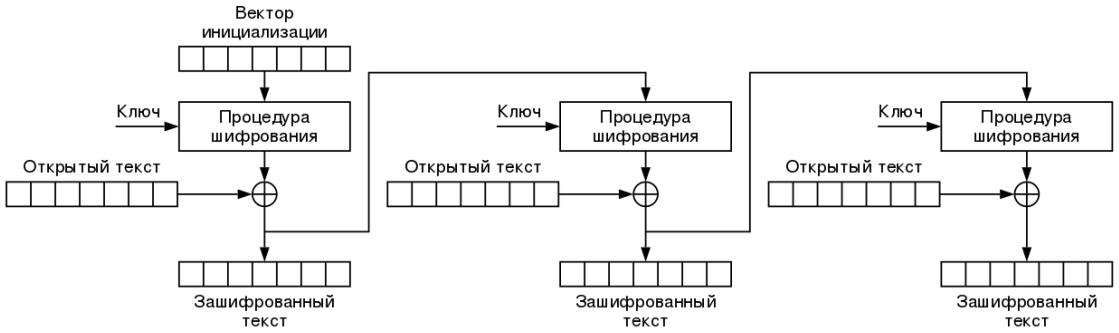

Рис. 3: Расшифрование в режиме обратной связи по шифротексту

дующие блоки.

Существует более сложный вариант использования режима, когда размер блока CFB не совпадет с размером блока шифра.

2. ОПИСАНИЕ ПРАКТИЧЕСКОЙ ЧАСТИ

Итак, моей задачей было программно реализовать алгоритм шифрования IDEA в режиме CFB. Для решения этой задачи, после изучения теоритической части, я выбрала язык программирования C++. В результате работы у меня получился класс IDEA (рис. 4).

Он имеет такие свойства как ключ (key), подключи (subKey), вектор инициаализации (feedback), режим работы (mode) и булевую переменную, отвечающую за режим шифрование/расшифрование (flag).

```
class IDEA {
private:
    vector<uint8_t> data;
    vector<uint8_t> key;
    vector<uint16_t> subKey;
    vector<uint8_t> feedback;
    MODE mode;
    bool flag;
public:
    IDEA(string key) { ...}
    IDEA(string key, string data) { ...}
    IDEA(string key, vector<uint8_t> data) { ...}
    IDEA(vector<uint8_t> key, vector<uint8_t> data) { ...}
    void setFeedBack(vector<uint8_t> feedback) { ...}
    void setFeedBack(string feedback) { ...}
    void setKey(string key) { ...}
    void setData(vector<uint8_t> data) { ...}
    void setData(string data) { ...}
    void generateSubkeys();
    void invertSubkeys();
    void encrypt(MODE mode, bool flag);
    vector<uint8_t> getDATA() {return data;}
    vector<uint8_t> getKey() {return key;}
    vector<uint16_t> getSubKey() {return subKey;}

//    void crypt(vector<uint8_t>& data);
    void crypt(vector<uint8_t>& data, size_t i);

    void cfb();
    void ecb();
    void ofb();
    void cbc();
};
```

Рис. 4: class IDEA.

Также имеет несколько конструкторов для удобства пользователя и методы, отвечающие за режимы шифрования.
Рассмотрим некоторые из них. В конструктуре вызывается метод *void generateSubkeys*(); (рис. 5), который генерирует 52 подключа. Сначала идет проверка, что ключ у нас состоит ровно из 128 бит. Потом, в первом цикле, мы проходимся по нему и склеиваем пару по 8 бит, тем самым деля ключ на 8 подключей, и получаем подключ в длинною в 16 бит. Во втором цикле мы генерируем оставшиеся 44 подключа, не забывая сдигать ключ через каждые 8 подключей на 25 бит влево.
Далее идем в метод *void cfb*() (рис. 6). Он отвечает за работу алгоритма в режиме CFB. Как мы можем видеть из кода сначала мы шифруем наш инициализированный вектор, затем складываем его по модулю 2 с блоком открытого текста. После этого у нас получается зашифрованный блок текса, также мы присваиваем его *feedback* и проходимся по циклу заново, пока не зашифруем весь текст. А если нам нужно расшифровать текст, то в алгоритме появляется небольшое изменение - мы сохраняем блок зашифрованного текста в переменную *clone* перед операцией *XOR*, а потом эти сохраненные

данные присваиваем $feedback$.

```
void IDEA::generateSubkeys() {
    if (key.size() != KEY_SIZE) {
        throw std::runtime_error("Error: Key must be 128 bits in length.");
    }

    uint16_t b1, b2;
    for (size_t i = 0; i < key.size() / 2; i++) {
        subKey[i] = glue2Bytes(key[2 * i], key[2 * i + 1]);
    }

    for (size_t i = key.size() / 2; i < subKey.size(); i++) {
        b1 = subKey[(i + 1) % 8 != 0 ? i - 7 : i - 15] << 9;
        b2 = subKey[(i + 2) % 8 < 2 ? i - 14 : i - 6] >> 7;
        subKey[i] = (b1 | b2) & 0xFFFF;
    }
}
```

Рис. 5: Генерация подключей.

```
void IDEA::cfb() {
    size_t i = 0;vector<uint8_t> clone(8);
    while(i < data.size()) {
        if(flag) {
            clone.clear();
            clone = vector<uint8_t>(data.begin()+i, data.begin()+i+8);
        }
        crypt(feedback, i);
        for(size_t j = 0; j < BLOCK_SIZE; j++) {
              data[i + j] ^= feedback[j];
              feedback[j] = data[i + j];
        }
        if(flag) {
            feedback.clear();
            feedback = clone;
        }
        i += 8;
    }

    cout << endl << " CFB:\n ";
    for(auto c: data) cout << c << " ";
}
```

Рис. 6: Режим CFB.

```
void IDEA::crypt(vector<uint8_t>& data, size_t i) {
    uint16_t d1 = glue2Bytes(data[i + 0], data[i + 1]);
    uint16_t d2 = glue2Bytes(data[i + 2], data[i + 3]);
    uint16_t d3 = glue2Bytes(data[i + 4], data[i + 5]);
    uint16_t d4 = glue2Bytes(data[i + 6], data[i + 7]);
    size_t j = 0;
    for (uint8_t round = 0; round < ROUNDS; round++) {
        uint16_t a = mult(d1, subKey[j++]);
        uint16_t b = add(d2, subKey[j++]);
        uint16_t c = add(d3, subKey[j++]);
        uint16_t d = mult(d4, subKey[j++]);
        uint16_t e = a ^ c;
        uint16_t f = b ^ d;

        uint16_t g = mult(e, subKey[j++]);
        uint16_t s = add(f, g);
        uint16_t h = mult(s, subKey[j++]);
        uint16_t k = add(g, h);

        d1 = a ^ h;
        d2 = c ^ h;
        d3 = b ^ k;
        d4 = d ^ k;
    }
    uint16_t res1 = mult(d1, subKey[j++]);
    uint16_t res2 = add(d3, subKey[j++]);
    uint16_t res3 = add(d2, subKey[j++]);
    uint16_t res4 = mult(d4, subKey[j]);

    data[i + 0] = static_cast<uint8_t>(res1 >> 8);
    data[i + 1] = static_cast<uint8_t>(res1);
    data[i + 2] = static_cast<uint8_t>(res2 >> 8);
    data[i + 3] = static_cast<uint8_t>(res2);
    data[i + 4] = static_cast<uint8_t>(res3 >> 8);
    data[i + 5] = static_cast<uint8_t>(res3);
    data[i + 6] = static_cast<uint8_t>(res4 >> 8);
    data[i + 7] = static_cast<uint8_t>(res4);
}
```

Рис. 7: Программная схема шифрования.

И рассмотрим последний, основной метод *void crypt()*, отвечающий за шифрование блока из 64-х бит. Здесь мы снова можем наблюдать за разбиением текса из 64 бит на 4 части по 16 бит методом склеивания пары из 8 бит в 16 бит. Далее происходит все то, что было описано в разделе 1.5 теоритической части. В конце происходит скливание 4 частей обратно в 64-х битный блок.

3. ЗАКЛЮЧЕНИЕ

Симметричный блочный алгоритм шифрования данных IDEA был в полной мере изучен с теоретической стороны, а также программно реализован на C++ в режиме обратной связи по шифротексту.

4. СПИСОК ЛИТЕРАТУРЫ

- Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си = Applied Cryptography. Protocols, Algorithms and Source Code in C
- Введение в криптографию / Под. Общ. Ред. В.В. Ященко — 3е изд., доп. М.: МЦНМО: «ЧеРо», 2000. 288 с.
- Алферов А.П., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В. Основы криптографии. Учебное пособие, 2е изд. испр. и доп. - М.:Гелиос АРВ, 2002. 480 с.: ил.
- Miss in the Middle Attacks on IDEA and Khufu. E. Biham, A. Biryukov, A. Shamir. 1999.
- https://ru.wikipedia.org/wiki/IDEA/Применение IDEA

5. ПРИЛОЖЕНИЕ

Окружение ОС:

OS: Arch Linux on Windows 10 x86_64
Kernel: 5.10.60.1-microsoft-standart-WSL2

Версия C++:

GCC 11.2, libstdc++6-dev, g++, make

Исходные коды программы, а также исходные коды отчета можно найти на гитхабе автора: FreeSoul777/IDEA