Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2

По дисциплине «Методы защиты информации»

По теме «Симметричная криптография. СТБ 34.101.31-2011»

Выполнил:

Студент гр. 653501

Никитинская А. С.

Проверил:

Артемьев В. С.

Минск 2019

Содержание

[1. Постановка задачи 3](#_Toc20490052)

[2. Краткие теоретические сведения 4](#_Toc20490053)

[3. Блок-схемы алгоритма 10](#_Toc20490057)

[Вывод 11](#_Toc20490058)

[Приложение 1. Исходный код программы 12](#_Toc20490059)

[Приложение 2. Скриншот работы программы 19](#_Toc20490060)

## 1. Постановка задачи

1) Изучить теоретические сведения.

2) Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма СТБ 34.101.31-2011 в различных режимах.

## Краткие теоретические сведения

СТБ 34.101.31-2011 «Информационные технологии и безопасность. Криптографические алгоритмы шифрования и контроля целостности» — государственный стандарт симметричного шифрования и контроля целостности Республики Беларусь. Принят в качестве стандарта симметричного шифрования республики Беларусь в 2011 году

СТБ 34.101.31-2011 определяет семейство криптографических алгоритмов, предназначенных для обеспечения конфиденциальности и контроля целостности данных. Обрабатываемыми данными являются двоичные слова (сообщения). Криптографические алгоритмы стандарта построены на основе базовых алгоритмов шифрования блока данных.

2.1 Назначение алгоритма СТБ

Настоящий предстандарт определяет семейство криптографических алгоритмов, предназначенных для обеспечения конфиденциальности и контроля целостности данных. Обрабатываемыми данными являются двоичные слова. Криптографические алгоритмы предстандарта построены на основе вспомогательных алгоритмов шифрования блоков данных двоичных слов длины 128. Криптографические алгоритмы делятся на шесть групп:

1) алгоритмы шифрования в режиме простой замены;

2) алгоритмы шифрования в режиме сцепления блоков;

3) алгоритмы шифрования в режиме гаммирования с обратной связью;

4) алгоритмы шифрования в режиме счетчика;

5) алгоритм выработки имитовставки;

6) алгоритм хэширования.

Первые четыре группы предназначены для обеспечения конфиденциальности. Каждая группа включает алгоритм зашифрования и алгоритм расшифрования на секретном ключе. Алгоритм зашифрования преобразует открытые данные в защищенные, а алгоритм расшифрования выполняет обратное преобразование. Стороны, располагающие общим секретным ключом, могут организовать защищенный обмен данными путем их зашифрования перед отправкой и расшифрования после получения.

Пятый алгоритм предназначен для создания имитовставок контрольных данных, которые определяются с использованием ключа. Стороны, располагающие общим секретным ключом, могут организовать контроль целостности при передаче данных путем добавления к ним имитовставок при отправке и проверки имитовставок при получении.

Проверка имитовставок дополнительно позволяет стороне-получателю убедиться в знании стороной-отправителем секретного ключа. Последний алгоритм предназначен для вычисления хэш-значений контрольных данных, которые определяются без использования ключа. Стороны могут организовать контроль целостности данных путем сравнения их хэш-значений с достоверными контрольными хэш-значениями. Изменение двоичного слова с высокой вероятностью приводит к изменению соответствующего хэш-значения и поэтому хэш-значения могут использоваться вместо самих слов, например в системах электронной цифровой подписи.

2.2 Ключ

В алгоритмах шифрования и выработки имитовставки используется ключ θ∈{0,1}256, который однозначно определяет криптографическое преобразование данных. Ключ должен вырабатываться без возможности предсказания, распространяться с

соблюдением мер конфиденциальности и храниться в секрете. Один и тот же ключ не должен использоваться в алгоритмах различных групп. Разрешается использовать ключ θ длины 256, который является результатом расширения короткого ключа длины 128 или 192. Пусть короткий ключ имеет вид θ1‖θ2‖. . .‖θd, где θi∈ {0,1}32 и d= 4 или d= 6. Процедура расширения состоят в определении слов θd+1, . . . , θ8∈ {0,1}32 с последующим формированием θ=θ1‖θ2‖. . .‖θ8. Должны использоваться следующие правила расширения:

1) при d= 4 установить:θ5←θ1,θ6←θ2,θ7←θ3,θ8←θ4;

2) при d= 6 установить:θ7←θ1⊕θ2⊕θ3,θ8←θ4⊕θ5⊕θ6.

2.3 Синхропосылка

При шифровании в режимах сцепления блоков, гаммирования с обратной связью и счетчика используется синхропосылка S∈ {0,1}128, которая обеспечивает уникальность криптографических преобразований на фиксированном ключе.

Синхропосылка не является секретным параметром, может добавляться к зашифрованным данным и передаваться вместе с ними. При зашифровании на одном и том же ключе должны использоваться различные синхропосылки.

2.4 Имитовставка

Имитовставкой слова X∈ {0,1}∗ на ключе θ∈ {0,1}256 является слово Y∈ {0,1}64. Разрешается использовать не все 64 символа имитовставки, а только первые l≤64 символов. При выборе l следует учитывать, что вероятность угадывания имитовставки заданного слова без знания ключа равняется 2−l.

2.5 Шифрование блоков данных

2.5.1 Входные и выходные данные

Входными данными алгоритмов зашифрования и расшифрования являются слово X∈{0,1}128 и ключ θ∈{0,1}256. Выходными данными является слово Y∈ {0,1}128, результат зашифрования либо расшифрования слова X на ключе θ:Y=Fθ(X) либо Y=F−1θ(X).

Входные данные подготавливаются следующим образом:

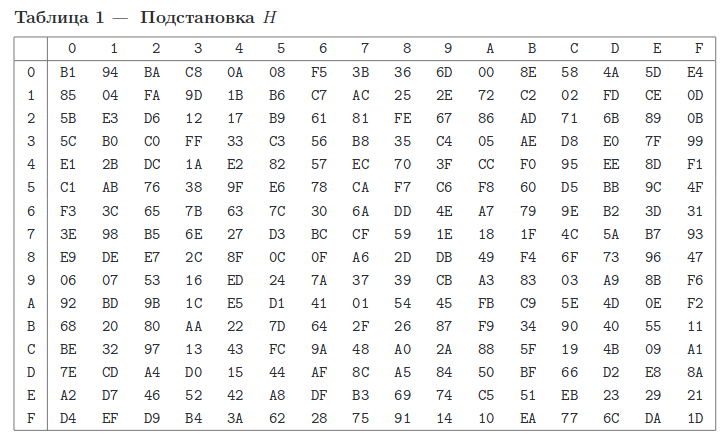
1) Слово X записывается в виде X=X1‖X2‖X3‖X4, где Xi∈{0,1}32.

2) Ключ θ записывается в виде θ=θ1‖θ2‖. . .‖θ8,θi∈ {0,1}32, и определяются тактовые ключи K1=θ1, K2=θ2, . . . , K8=θ8, K9=θ1, K10=θ2 , . . . , K56=θ8.

2.5.2 Переменные и вспомогательные преобразования

Переменные. Используются переменные a, b, c, d, e со значениями из {0,1}32.

Подстановка H. Подстановка H:{0,1}8→{0,1}8 задается таблицей 1. В таблице 1 используется шестнадцатеричное представление слов u ∈{0,1}8. Если u=IJ16, то значение H(u) находится на пересечении строки I и столбца J. Например, H(A216) =9B16.



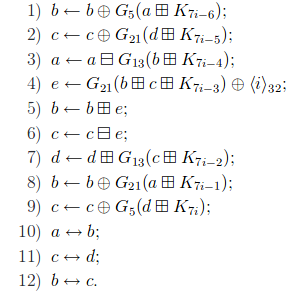
Преобразования Gr(r= 5,13,21). Преобразование Gr :{0,1}32→ {0,1}32 ставит в соответствие слову u=u1‖u2‖u3‖u4,ui∈{0,1}8, слово Gr(u) =λr(H(u1)‖H(u2)‖H(u3)‖H(u4)).

2.5.3 Алгоритм зашифрования

Для зашифрования слова X на ключе θ выполняются следующие шаги:

1 Установить a←X1,b←X2,c←X3,d←X4.

2 Для i= 1,2, . . . ,8 выполнить



3 Установить Y←b‖d‖a‖c.

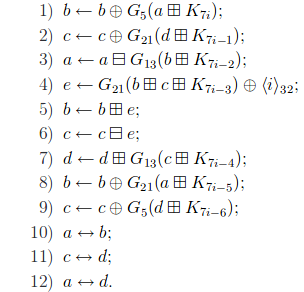
4 Возвратить Y.

2.5.4 Алгоритм расшифрования

Для расшифрования слова X на ключе θ выполняются следующие шаги:

1 Установить a←X1,←X2,c←X3,d←X4.

2 Для i= 8,7, . . . ,1 выполнить:



3 Установить Y←c‖a‖d‖b.

4 Возвратить Y.

2.5.5 Шифрование в режиме простой замены

Входными данными алгоритмов зашифрования и расшифрования являются слово X ∈{0,1}128∗ и ключ θ∈{0,1}256. Выходными данными является слово Y∈ {0,1}|X| результата зашифрования либо расшифрования X на ключе θ. Входное слово X записывается в виде X=X1‖X2‖. . .‖Xd, Xi∈{0,1}128. При шифровании словам Xi ставятся в соответствие слова Yi∈ {0,1}128 , из которых затем составляется Y.

Зашифрование слова X на ключе θ состоит в выполнении следующих шагов:

1 Для i= 1,2, . . . , d выполнить Yi←Fθ(Xi).

2 Установить Y←Y1‖Y2‖. . .‖Yd.

3 Возвратить Y.

Расшифрование слова X на ключе θ состоит в выполнении следующих шагов:

1 Для i= 1,2, . . . , d выполнить Yi←F−1θ(Xi).

2 Установить Y←Y1‖Y2‖. . .‖Yd.

3 Возвратить Y.

2.5.6 Шифрование в режиме сцепления блоков

Входными данными алгоритмов зашифрования и расшифрования являются слово X∈{0,1}128∗, ключ θ∈{0,1}256 и синхропосылка S∈{0,1}128. Выходными данными является слово Y∈ {0,1}|X| результат зашифрования либо расшифрования X на ключе θ при использовании синхропосылки S . Входное слово X записывается в виде X=X1‖X2‖. . .‖Xd, Xi∈{0,1}128. При шифровании словам Xi ставятся в соответствие слова Yi∈ {0,1}128, из которых затем составляется Y.

Зашифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1 Для i= 1,2, . . . , d выполнить Yi←Fθ(Xi⊕Yi−1), где Y0=Fθ(S).

2 Установить Y←Y1‖Y2‖. . .‖Yd.

3 Возвратить Y.

Расшифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1 Для i= 1,2, . . . , d выполнить Yi←F−1θ(Xi)⊕Xi−1, где X0=Fθ(S).

2 Установить Y←Y1‖Y2‖. . .‖Yd.

3 Возвратить Y.

2.5.7 Шифрование в режиме гаммирования с обратной связью

Входными данными алгоритмов зашифрования и расшифрования являются слово X∈{0,1}∗, ключ θ∈{0,1}256 и синхропосылка S∈{0,1}128. Выходными данными является слово Y∈ {0,1}|X| результат зашифрования либо расшифрования X на ключе θ при использовании синхропосылки S .Входное слово X записывается в виде X=X1‖X2‖. . .‖Xd,|X1|=|X2|=. . .=|Xd−1|= 128,|Xd|≤128. При шифровании словам Xi ставятся в соответствие слова Yi∈{0,1}|Xi|, из которых затем составляется Y.

Зашифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1 Для i= 1,2, . . . , d выполнить Yi←Xi⊕L|Xi|(Fθ(Yi−1)), гдеY0=S.

2 Установить Y←Y1‖Y2‖. . .‖Yd.

3 Возвратить Y.

Расшифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1 Для i= 1,2, . . . , d выполнить Yi←Xi⊕L|Xi|(Fθ(Xi−1)), где X0=S.

2 Установить Y←Y1‖Y2‖. . .‖Yd.

3 Возвратить Y.

2.5.8 Шифрование в режиме счетчика

Входными данными алгоритмов зашифрования и расшифрования являются слово X∈{0,1}∗, ключ θ∈{0,1}256 и синхропосылка S∈{0,1}128.Выходными данными является слово Y∈ {0,1}|X| результат зашифрования либо расшифрования X на ключе θ при использовании синхропосылки S. Входное слово X записывается в виде X=X1‖X2‖. . .‖Xd,|X1|=|X2|=. . .=|Xd−1|= 128,|Xd|≤128.При шифровании словам Xi ставятся в соответствие слова Yi∈{0,1}|Xi|, из которых затем составляется Y.

Используется переменная s со значениями из {0,1}128.

Зашифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1 Установить s←Fθ(S).

2 Для i= 1,2, . . . , d выполнить:

1) s←s¢〈1〉128,

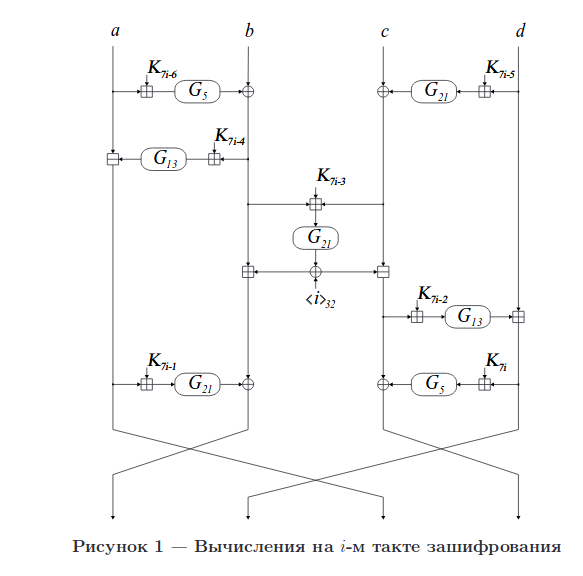
2)Yi←Xi⊕L|Xi|(Fθ(s)).

3 Установить Y←Y1‖Y2‖. . .‖Yd.

4 Возвратить Y.

Расшифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении тех же шагов, что и при зашифровании.

## 3. Блок-схема алгоритма



## Вывод

В ходе написания лабораторной работы были изучены алгоритмы шифрования и дешифрования СТБ, а также написаны их программные реализации. Были получены навыки усложнения и увеличения криптостойкости алгоритма СТБ, а также изучены модификации и режимы работы алгоритма СТБ.

## Приложение 1. Исходный код программы

const vector<vector<int>> H\_TABLE = {

{0xB1, 0x94, 0xBA, 0xC8, 0x0A, 0x08, 0xF5, 0x3B, 0x36, 0x6D, 0x00, 0x8E, 0x58, 0x4A, 0x5D, 0xE4},

{0x85, 0x04, 0xFA, 0x9D, 0x1B, 0xB6, 0xC7, 0xAC, 0x25, 0x2E, 0x72, 0xC2, 0x02, 0xFD, 0xCE, 0x0D},

{0x5B, 0xE3, 0xD6, 0x12, 0x17, 0xB9, 0x61, 0x81, 0xFE, 0x67, 0x86, 0xAD, 0x71, 0x6B, 0x89, 0x0B},

{0x5C, 0xB0, 0xC0, 0xFF, 0x33, 0xC3, 0x56, 0xB8, 0x35, 0xC4, 0x05, 0xAE, 0xD8, 0xE0, 0x7F, 0x99},

{0xE1, 0x2B, 0xDC, 0x1A, 0xE2, 0x82, 0x57, 0xEC, 0x70, 0x3F, 0xCC, 0xF0, 0x95, 0xEE, 0x8D, 0xF1},

{0xC1, 0xAB, 0x76, 0x38, 0x9F, 0xE6, 0x78, 0xCA, 0xF7, 0xC6, 0xF8, 0x60, 0xD5, 0xBB, 0x9C, 0x4F},

{0xF3, 0x3C, 0x65, 0x7B, 0x63, 0x7C, 0x30, 0x6A, 0xDD, 0x4E, 0xA7, 0x79, 0x9E, 0xB2, 0x3D, 0x31},

{0x3E, 0x98, 0xB5, 0x6E, 0x27, 0xD3, 0xBC, 0xCF, 0x59, 0x1E, 0x18, 0x1F, 0x4C, 0x5A, 0xB7, 0x93},

{0xE9, 0xDE, 0xE7, 0x2C, 0x8F, 0x0C, 0x0F, 0xA6, 0x2D, 0xDB, 0x49, 0xF4, 0x6F, 0x73, 0x96, 0x47},

{0x06, 0x07, 0x53, 0x16, 0xED, 0x24, 0x7A, 0x37, 0x39, 0xCB, 0xA3, 0x83, 0x03, 0xA9, 0x8B, 0xF6},

{0x92, 0xBD, 0x9B, 0x1C, 0xE5, 0xD1, 0x41, 0x01, 0x54, 0x45, 0xFB, 0xC9, 0x5E, 0x4D, 0x0E, 0xF2},

{0x68, 0x20, 0x80, 0xAA, 0x22, 0x7D, 0x64, 0x2F, 0x26, 0x87, 0xF9, 0x34, 0x90, 0x40, 0x55, 0x11},

{0xBE, 0x32, 0x97, 0x13, 0x43, 0xFC, 0x9A, 0x48, 0xA0, 0x2A, 0x88, 0x5F, 0x19, 0x4B, 0x09, 0xA1},

{0x7E, 0xCD, 0xA4, 0xD0, 0x15, 0x44, 0xAF, 0x8C, 0xA5, 0x84, 0x50, 0xBF, 0x66, 0xD2, 0xE8, 0x8A},

{0xA2, 0xD7, 0x46, 0x52, 0x42, 0xA8, 0xDF, 0xB3, 0x69, 0x74, 0xC5, 0x51, 0xEB, 0x23, 0x29, 0x21},

{0xD4, 0xEF, 0xD9, 0xB4, 0x3A, 0x62, 0x28, 0x75, 0x91, 0x14, 0x10, 0xEA, 0x77, 0x6C, 0xDA, 0x1D}};

class Stb {

private:

string password;

string text;

vector<vector<int>> sub\_keys;

static long long get\_data\_from\_H\_table(string val) {

if (val[0] != '0' && val[1] != 'x') {

val = Utils::string\_slice(val, 2);

}

while ((val[0] == '0' || val[0] == 'x') && val.size() > 2) {

val = Utils::string\_slice(val, 1);

}

if (val.size() > 2) {

throw HTableOutOfRange();

} else if (val.size() == 1) {

auto i = std::stoi("0x0", nullptr, 16);

auto j = std::stoi(val, nullptr, 16);

return H\_TABLE[i][j];

}

vector<string> data = Utils::n\_split\_string(val, 1);

auto i = std::stoi(data[0], nullptr, 16);

auto j = std::stoi(data[1], nullptr, 16);

return H\_TABLE[i][j];

}

static vector<int> plus\_mod\_32(vector<int> a, vector<int> b) {

auto int\_a = Utils::byte\_to\_long(a);

auto int\_b = Utils::byte\_to\_long(b);

auto int\_result = Utils::inf\_int\_to\_long\_num(Utils::add\_inf\_int(int\_a, int\_b) % 4294967296);

auto result = Utils::long\_long\_to\_bin\_value\_num(int\_result, 32);

return result;

}

static vector<int> minus\_mod\_32(vector<int> a, vector<int> b) {

auto int\_a = Utils::byte\_to\_long(a);

auto int\_b = Utils::byte\_to\_long(b);

auto int\_result = Utils::inf\_int\_to\_long\_num(Utils::sub\_inf\_int(int\_a, int\_b) % 4294967296);

if (int\_result < 0) {

int\_result = 4294967296 + int\_result;

}

auto result = Utils::long\_long\_to\_bin\_value\_num(int\_result, 32);

return result;

}

static vector<int> g\_transition(vector<int> data, int num) {

auto int\_data = Utils::n\_split(data, 8);

auto new\_data = vector<int>();

for (const auto &val : int\_data) {

auto int\_val = Utils::byte\_to\_int(val);

auto hex\_val = Utils::int\_to\_hex(int\_val);

auto H = Stb::get\_data\_from\_H\_table(hex\_val);

new\_data.push\_back(H);

}

auto new\_data\_bin = vector<int>();

for (auto i = 0; i < new\_data.size(); i++) {

auto temp = Utils::int\_bin\_value\_num(new\_data[i], 8);

for (auto j = 0; j < temp.size(); j++) {

new\_data\_bin.push\_back(temp[j]);

}

}

auto int\_s = Utils::byte\_to\_long(new\_data\_bin);

int\_s = ((int\_s >> (32 - num)) | (int\_s << num)) & 0xFFFFFFFF;

auto result = Utils::long\_long\_to\_bin\_value\_num(int\_s, 32);

return result;

}

public:

Stb(string password, string text) {

this->password = std::move(password);

this->text = std::move(text);

auto encode\_key = Utils::string\_to\_bit\_array(this->password);

this->sub\_keys = Utils::n\_split(encode\_key, 32);

}

string run\_encode() {

auto raw\_text\_bits = Utils::string\_to\_bit\_array(this->text);

if (raw\_text\_bits.size() != 128) {

throw InvalidTextBlockSize(128);

}

auto raw\_splitted\_text = Utils::n\_split(raw\_text\_bits, 32);

auto a = raw\_splitted\_text[0];

auto b = raw\_splitted\_text[1];

auto c = raw\_splitted\_text[2];

auto d = raw\_splitted\_text[3];

auto all\_keys = vector<vector<int>>();

for (auto i = 0; i < 7; i++) {

auto copy = Utils::vector\_copy(this->sub\_keys);

for (const auto &j : copy) {

all\_keys.push\_back(j);

}

}

for (auto i = 1; i < 9; i++) {

auto sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 6 - 1]);

auto temp = Stb::plus\_mod\_32(a, sub\_key);

temp = Stb::g\_transition(temp, 5);

b = Utils::vector\_xor(b, temp);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 5 - 1]);

temp = Stb::plus\_mod\_32(d, sub\_key);

temp = Stb::g\_transition(temp, 21);

c = Utils::vector\_xor(c, temp);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 4 - 1]);

temp = Stb::plus\_mod\_32(b, sub\_key);

temp = Stb::g\_transition(temp, 13);

a = Stb::minus\_mod\_32(a, temp);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 3 - 1]);

temp = Stb::plus\_mod\_32(b, c);

temp = Stb::plus\_mod\_32(temp, sub\_key);

temp = Stb::g\_transition(temp, 21);

auto val\_i\_int = i % 4294967296;

auto val\_i = Utils::long\_long\_to\_bin\_value\_num(val\_i\_int, 32);

auto e = Utils::vector\_xor(temp, val\_i);

b = Stb::plus\_mod\_32(b, e);

c = Stb::minus\_mod\_32(c, e);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 2 - 1]);

temp = Stb::plus\_mod\_32(c, sub\_key);

temp = Stb::g\_transition(temp, 13);

d = Stb::plus\_mod\_32(d, temp);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 1 - 1]);

temp = Stb::plus\_mod\_32(a, sub\_key);

temp = Stb::g\_transition(temp, 21);

b = Utils::vector\_xor(b, temp);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 1]);

temp = Stb::plus\_mod\_32(d, sub\_key);

temp = Stb::g\_transition(temp, 5);

c = Utils::vector\_xor(c, temp);

auto tmp = Utils::vector\_copy(a);

a = Utils::vector\_copy(b);

b = Utils::vector\_copy(tmp);

tmp = Utils::vector\_copy(d);

d = Utils::vector\_copy(c);

c = Utils::vector\_copy(tmp);

tmp = Utils::vector\_copy(b);

b = Utils::vector\_copy(c);

c = Utils::vector\_copy(tmp);

}

auto encoded\_text = Utils::vector\_concat(Utils::vector\_concat(b, d), Utils::vector\_concat(a, c));

auto result = Utils::bit\_array\_to\_string(encoded\_text);

return result;

}

string run\_decode() {

auto raw\_text\_bits = Utils::string\_to\_bit\_array(this->text);

if (raw\_text\_bits.size() != 128) {

throw InvalidTextBlockSize(128);

}

auto raw\_splitted\_text = Utils::n\_split(raw\_text\_bits, 32);

auto a = raw\_splitted\_text[0];

auto b = raw\_splitted\_text[1];

auto c = raw\_splitted\_text[2];

auto d = raw\_splitted\_text[3];

auto all\_keys = vector<vector<int>>();

for (auto i = 0; i < 7; i++) {

auto copy = Utils::vector\_copy(this->sub\_keys);

for (const auto &j : copy) {

all\_keys.push\_back(j);

}

}

for (auto i = 8; i > 0; i--) {

auto sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 1]);

auto temp = Stb::plus\_mod\_32(a, sub\_key);

temp = Stb::g\_transition(temp, 5);

b = Utils::vector\_xor(b, temp);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 1 - 1]);

temp = Stb::plus\_mod\_32(d, sub\_key);

temp = Stb::g\_transition(temp, 21);

c = Utils::vector\_xor(c, temp);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 2 - 1]);

temp = Stb::plus\_mod\_32(b, sub\_key);

temp = Stb::g\_transition(temp, 13);

a = Stb::minus\_mod\_32(a, temp);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 3 - 1]);

temp = Stb::plus\_mod\_32(b, c);

temp = Stb::plus\_mod\_32(temp, sub\_key);

temp = Stb::g\_transition(temp, 21);

auto val\_i\_int = i % 4294967296;

auto val\_i = Utils::long\_long\_to\_bin\_value\_num(val\_i\_int, 32);

auto e = Utils::vector\_xor(temp, val\_i);

b = Stb::plus\_mod\_32(b, e);

c = Stb::minus\_mod\_32(c, e);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 4 - 1]);

temp = Stb::plus\_mod\_32(c, sub\_key);

temp = Stb::g\_transition(temp, 13);

d = Stb::plus\_mod\_32(d, temp);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 5 - 1]);

temp = Stb::plus\_mod\_32(a, sub\_key);

temp = Stb::g\_transition(temp, 21);

b = Utils::vector\_xor(b, temp);

sub\_key = Utils::vector\_copy(all\_keys[7 \* i - 6 - 1]);

temp = Stb::plus\_mod\_32(d, sub\_key);

temp = Stb::g\_transition(temp, 5);

c = Utils::vector\_xor(c, temp);

auto tmp = Utils::vector\_copy(a);

a = Utils::vector\_copy(b);

b = Utils::vector\_copy(tmp);

tmp = Utils::vector\_copy(d);

d = Utils::vector\_copy(c);

c = Utils::vector\_copy(tmp);

tmp = Utils::vector\_copy(d);

d = Utils::vector\_copy(a);

a = Utils::vector\_copy(tmp);

}

auto decoded\_text = Utils::vector\_concat(Utils::vector\_concat(c, a), Utils::vector\_concat(d, b));

auto result = Utils::bit\_array\_to\_string(decoded\_text);

return result;

}

string run\_encode\_simple\_change() {

if (this->text.size() % 16 != 0) {

throw InvalidTextBlockSize(16);

}

auto encoded\_text = vector<string>();

auto raw\_text\_splitted = Utils::n\_split\_string(this->text, 16);

for (const auto & part: raw\_text\_splitted) {

encoded\_text.push\_back(Stb::encrypt(this->password, part));

}

return Utils::join\_strings(encoded\_text);

}

string run\_decode\_simple\_change() {

if (this->text.size() % 16 != 0) {

throw InvalidTextBlockSize(16);

}

auto decoded\_text = vector<string>();

auto raw\_text\_splitted = Utils::n\_split\_string(this->text, 16);

for (const auto & part: raw\_text\_splitted) {

decoded\_text.push\_back(Stb::decrypt(this->password, part));

}

return Utils::join\_strings(decoded\_text);

}

string run\_encode\_clutch\_blocks(string synchrosend) {

if (this->text.size() % 16 != 0) {

throw InvalidTextBlockSize(16);

}

auto encoded\_text = vector<string>();

auto raw\_text\_splitted = Utils::n\_split\_string(this->text, 16);

auto temp = Stb::encrypt(this->password, synchrosend);

for (const auto & part: raw\_text\_splitted) {

auto res = Utils::bit\_array\_to\_string(

Utils::vector\_xor(

Utils::string\_to\_bit\_array(temp),

Utils::string\_to\_bit\_array(part))

);

auto encoded\_part = Stb::encrypt(this->password, res);

temp = encoded\_part;

encoded\_text.push\_back(encoded\_part);

}

return Utils::join\_strings(encoded\_text);

}

string run\_decode\_clutch\_blocks(string synchrosend) {

if (this->text.size() % 16 != 0) {

throw InvalidTextBlockSize(16);

}

auto decoded\_text = vector<string>();

auto raw\_text\_splitted = Utils::n\_split\_string(this->text, 16);

auto temp = Stb::encrypt(this->password, synchrosend);

for (const auto & part: raw\_text\_splitted) {

auto res = Utils::bit\_array\_to\_string(

Utils::vector\_xor(

Utils::string\_to\_bit\_array(Stb::decrypt(this->password, part)),

Utils::string\_to\_bit\_array(temp))

);

decoded\_text.push\_back(res);

temp = part;

}

return Utils::join\_strings(decoded\_text);

}

string run\_encode\_gamma\_with\_feedback(string synchrosend) {

if (this->text.size() % 16 != 0) {

throw InvalidTextBlockSize(16);

}

auto encoded\_text = vector<string>();

auto raw\_text\_splitted = Utils::n\_split\_string(this->text, 16);

auto temp = synchrosend;

for (const auto & part: raw\_text\_splitted) {

auto enc\_temp = Stb::encrypt(this->password, temp);

auto enc\_sliced = Utils::string\_slice(enc\_temp, 0, part.size());

auto res = Utils::bit\_array\_to\_string(

Utils::vector\_xor(

Utils::string\_to\_bit\_array(part),

Utils::string\_to\_bit\_array(enc\_sliced))

);

encoded\_text.push\_back(res);

temp = res;

}

return Utils::join\_strings(encoded\_text);

}

string run\_decode\_gamma\_with\_feedback(string synchrosend) {

if (this->text.size() % 16 != 0) {

throw InvalidTextBlockSize(16);

}

auto decoded\_text = vector<string>();

auto raw\_text\_splitted = Utils::n\_split\_string(this->text, 16);

auto temp = synchrosend;

for (const auto & part: raw\_text\_splitted) {

auto enc\_temp = Stb::encrypt(this->password, temp);

auto enc\_sliced = Utils::string\_slice(enc\_temp, 0, part.size());

auto res = Utils::bit\_array\_to\_string(

Utils::vector\_xor(

Utils::string\_to\_bit\_array(part),

Utils::string\_to\_bit\_array(enc\_sliced))

);

decoded\_text.push\_back(res);

temp = part;

}

return Utils::join\_strings(decoded\_text);

}

string run\_encode\_with\_counter(string synchrosend) {

if (this->text.size() % 16 != 0) {

throw InvalidTextBlockSize(16);

}

auto encoded\_text = vector<string>();

auto raw\_text\_splitted = Utils::n\_split\_string(this->text, 16);

auto temp = Stb::encrypt(this->password, synchrosend);

auto ones = vector<int>();

for (auto i = 0; i < 128; i++) {

ones.push\_back(1);

}

for (const auto & part: raw\_text\_splitted) {

InfInt int\_a = Utils::byte\_to\_inf\_long(Utils::string\_to\_bit\_array(temp));

InfInt int\_b = Utils::byte\_to\_inf\_long(ones);

InfInt sum\_a\_b = int\_a + int\_b;

InfInt mod = sum\_a\_b % 4294967296;

auto int\_result = Utils::inf\_int\_to\_long\_num(mod);

auto plus\_mode = Utils::long\_long\_to\_bin\_value\_num(int\_result, 32);

auto s = vector<int>();

for (auto const & item: plus\_mode) {

auto byte = Utils::int\_bin\_value\_num(item, 8);

for (auto const & bit: byte) {

s.push\_back(bit);

}

if (s.size() >= 128) {

break;

}

}

temp = Utils::bit\_array\_to\_string(s);

temp = Stb::encrypt(this->password, temp);

auto sliced\_s = Utils::string\_slice(temp, 0, part.size());

auto res = Utils::bit\_array\_to\_string(

Utils::vector\_xor(

Utils::string\_to\_bit\_array(part),

Utils::string\_to\_bit\_array(sliced\_s))

);

encoded\_text.push\_back(res);

}

return Utils::join\_strings(encoded\_text);

}

string run\_decode\_with\_counter(string synchrosend) {

return Stb::encrypt\_with\_counter(this->password, this->text, synchrosend);

}

static string encrypt(string password, string text) {

return Stb(password, text).run\_encode();

}

static string decrypt(string password, string text) {

return Stb(password, text).run\_decode();

}

static string encrypt\_clutch\_blocks(string password, string text, string synchrosend) {

return Stb(password, text).run\_encode\_clutch\_blocks(synchrosend);

}

static string decrypt\_clutch\_blocks(string password, string text, string synchrosend) {

return Stb(password, text).run\_decode\_clutch\_blocks(synchrosend);

}

static string encrypt\_simple\_change(string password, string text) {

return Stb(password, text).run\_encode\_simple\_change();

}

static string decrypt\_simple\_change(string password, string text) {

return Stb(password, text).run\_decode\_simple\_change();

}

static string encrypt\_gamma\_with\_feedback(string password, string text, string synchrosend) {

return Stb(password, text).run\_encode\_gamma\_with\_feedback(synchrosend);

}

static string decrypt\_gamma\_with\_feedback(string password, string text, string synchrosend) {

return Stb(password, text).run\_decode\_gamma\_with\_feedback(synchrosend);

}

static string encrypt\_with\_counter(string password, string text, string synchrosend) {

return Stb(password, text).run\_encode\_with\_counter(synchrosend);

}

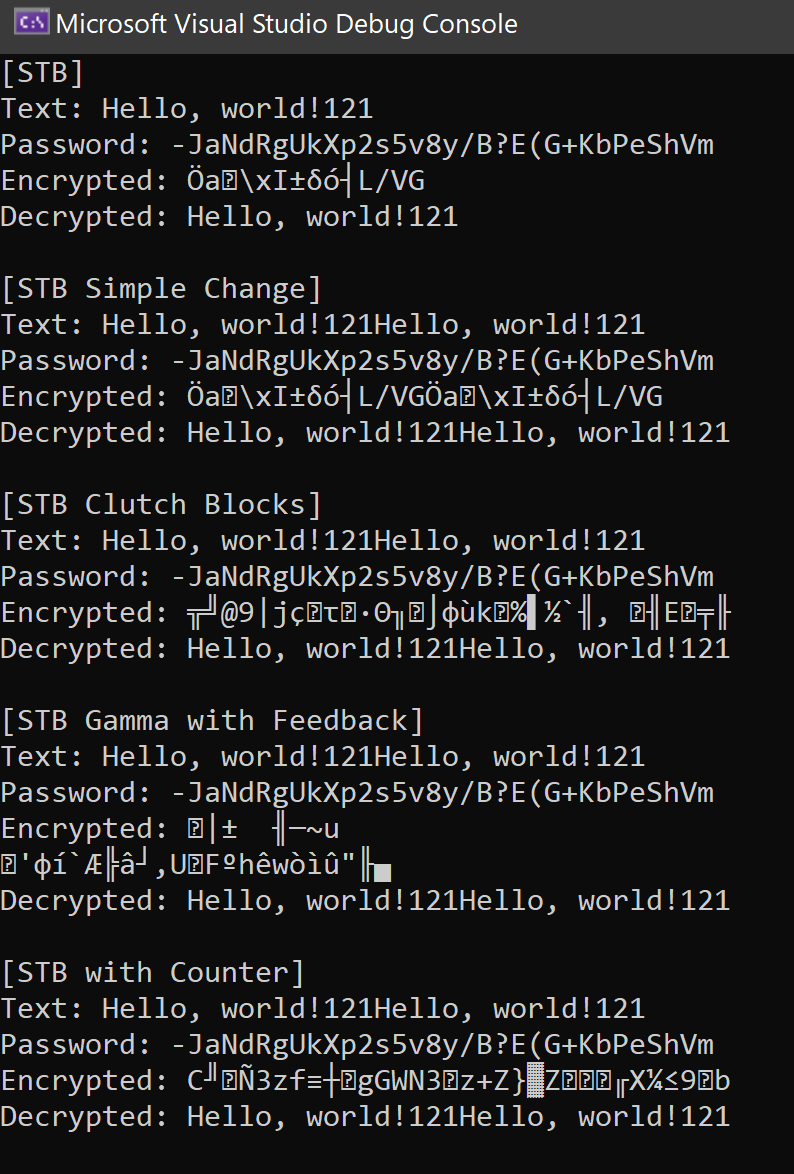
static string decrypt\_with\_counter(string password, string text, string synchrosend) {

return Stb(password, text).run\_decode\_with\_counter(synchrosend);

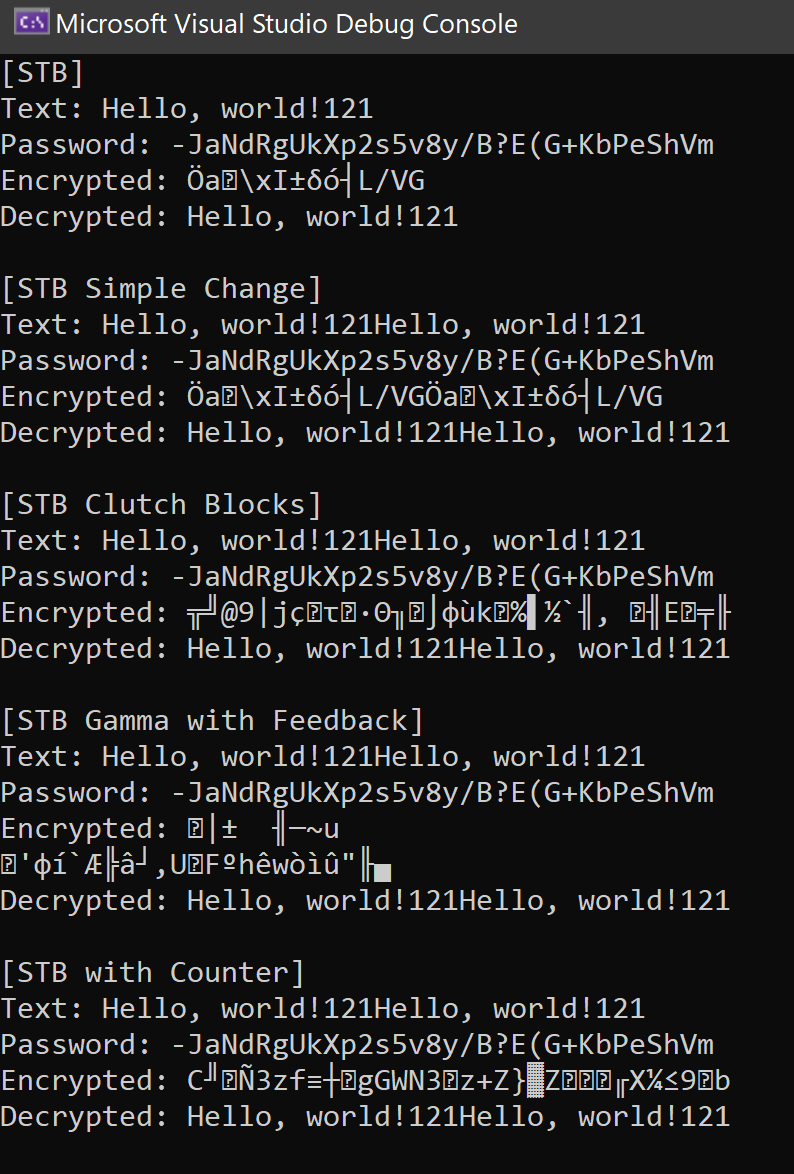
}

};

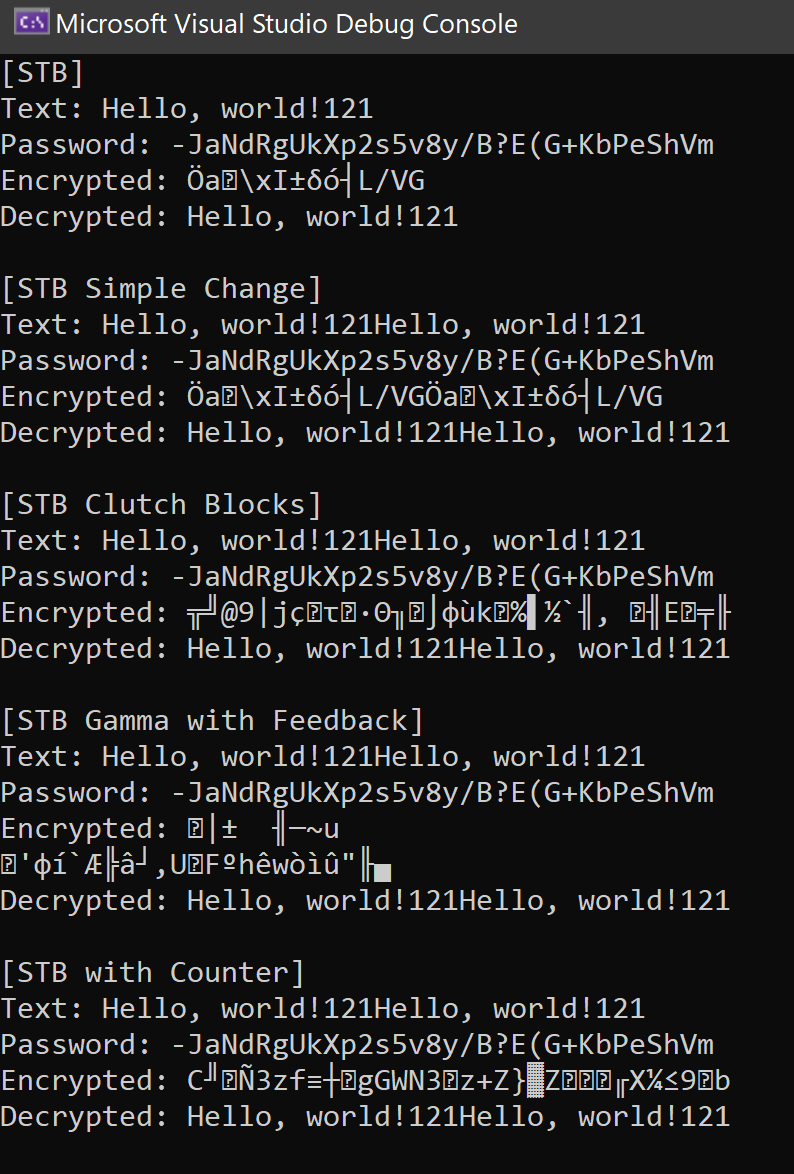
## Приложение 2. Скриншот работы программы



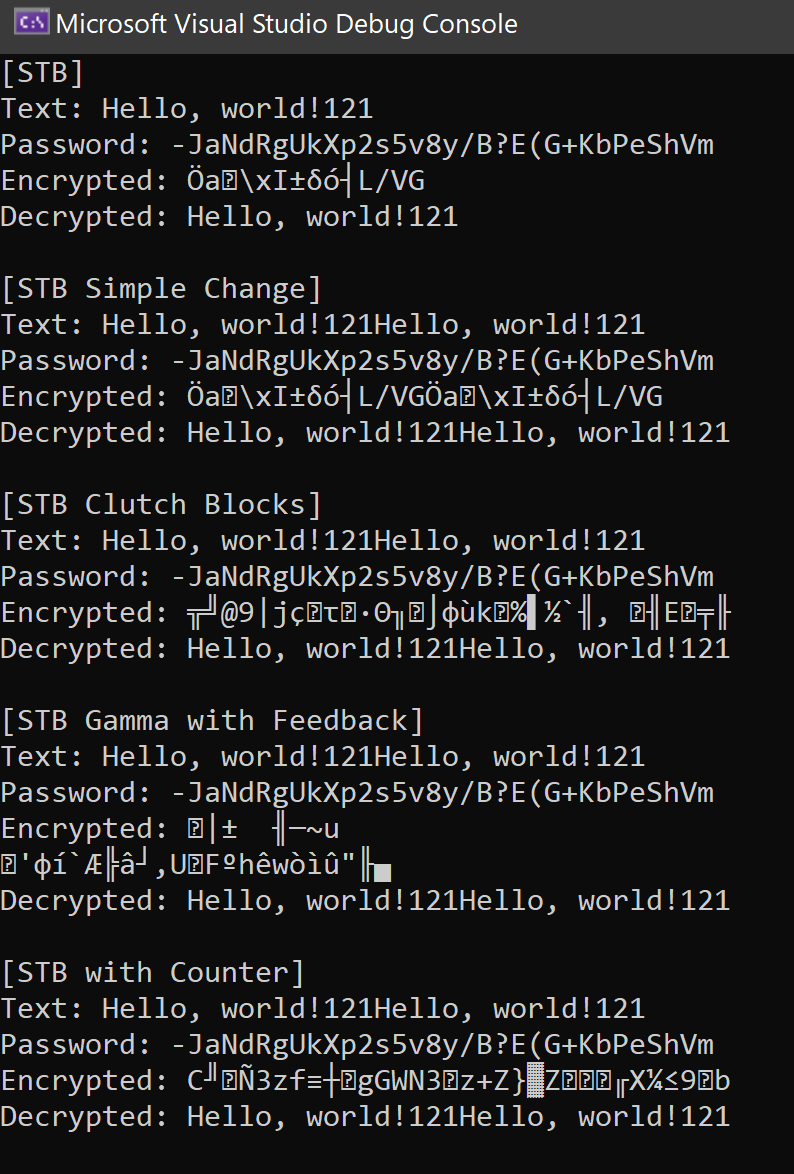
Шифрование в режиме счетчика



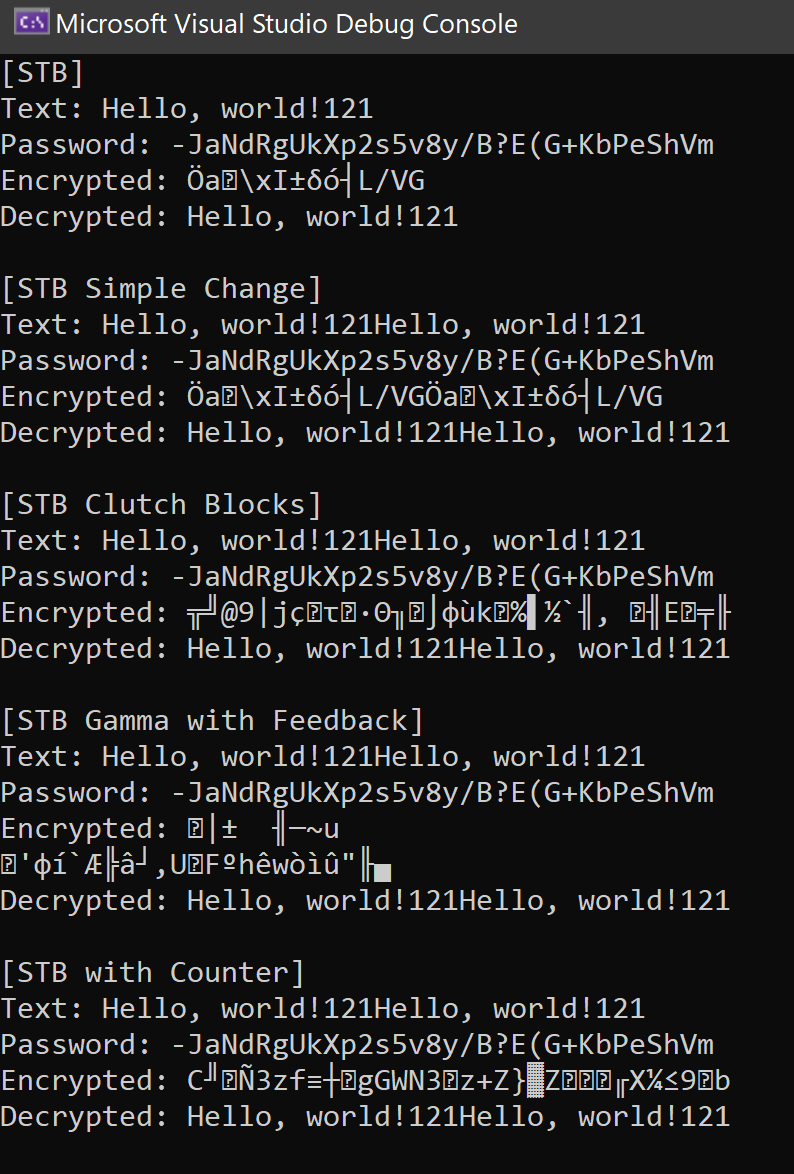
Шифрование в режиме гаммирования с обратной связью



Шифрование в режиме сцепления блоков



Шифрование в режиме простой замены



Шифрование