МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»
КАФЕДРА №51

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доц., к.т.н.		Окатов А.В
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

СИНХРОННОЕ ГАММИРОВАНИЕ

по курсу: Программно-аппаратные средства защиты информации в инфокоммуникационных системах связи

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	5711М		Пятаков В.С.
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2017

Цель работы

Изучение потоковых шифров на примере синхронного гаммирования.

Порядок выполнения работы

1. Реализовать синхронное гаммирование.
2. Провести тест на решетчатость для параметров алгоритма .
3. Построить функцию автокорреляции для параметров алгоритма.
4. Внести непредсказуемость путем внесения изменения в работу узлов шифрования на каком либо такте работы шифратора.
5. Оценить сложность взлома , методом полного перебора .
6. Сделать выводы по полученным результатам в ходе исследования в..

Описание алгоритма шифрования

Характерной особенностью поточных шифров есть побитная обработка информации. При этом шифрование и дешифрование может обрывается в произвольный момент времени. И как только связь восстановлена можно продолжать процедуру без проблем.

Шифрование происходит путем логических операций над битом ключа и битом исходного текста. Это происходит по тому, что сколько бы мы не создавали шифрующих битов, все равно накладываться будет один бит шифрующего на один бит исходного путем комбинации из функций XOR и отрицаний.

Классический пример шифра Вернама показана на рис.1. Такие шифры отлично подходят для локальный сетей, где решают часть проблем защиты информации в сетях.

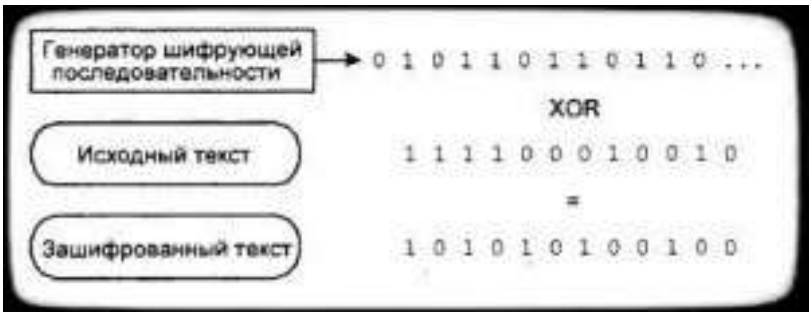

Рисунок 1 - шифра Вернама

Поточные шифры называют шифры гаммирования. Также само шифрование есть методом защиты информации. Эти шифры в разы быстрее своих конкурентов —

блочных шифров, если оно реализовано аппаратно. Если же реализация программная, здесь скорость может быть даже меньше блоковых шифров.

Функция которая формирует гамму, руководствуется трёма компонентами:

- ключ;
- номер текущего шага шифрования;
- ближние биты исходного или зашифрованного текста от текущей позиции;

Если схема формирования гаммы и ключ не секретные, то такой шифр превращается в скремблер. Скремблеры много используются в системах связи для повышения характеристик транслируемого сигнала. Частота появления нулей и единиц в таком сигнале близка к 0,5.

Зависимость от номера текущей позиции, если и присутствует в функции то пассивно. Просто на каждом шаге шифрования производятся однотипные преобразования ,где в любом случае используют текущий номер в общем потоке гаммы. Включения в функцию бит ближний бит исходного или шифрованного потока увеличивает криптостойкость шифра но при этом уменьшает скорость шифрования.

Шаблон зависимости основных свойств потоковых шифров показана на рис.2. Это только шаблон, нужно всегда учитывать конкретную ситуацию.

	Гамма зависит от бит исходного или зашифрованного текста	**Гамма НЕ зависит от бит исходного или зашифрованного текста**
Гамма зависит от номера текущего такта шифрования	(–) дешифратор теряет синхронизацию при ошибке "вставка/пропуск бита" в канале связи (–) дешифратор размножает ошибки "искажение бита" в канале связи (+) схема устойчива к атаке по известному исходному тексту	(–) дешифратор теряет синхронизацию при ошибке "вставка/пропуск бита" в канале связи (+) дешифратор не размножает ошибки "искажение бита" в канале связи (+) схема устойчива к атаке по известному исходному тексту
Гамма НЕ зависит от номера текущего такта шифрования	(+) дешифратор не теряет синхронизацию при ошибке класса "вставка/пропуск бита" в канале связи (–) дешифратор размножает ошибки класса "искажение бита" в канале связи (–) схема не устойчива к атаке по известному исходному тексту	

Рисунок 2 - Шаблон зависимости основных свойств потоковых шифров

Стойкость шифров гаммирования

Атаки на такие шифры, основаны на статистических особенностях гаммы от примера со случайным потоком гаммы, или на повторяющие участки шифрования гаммой.

Первый случай нужно учитывать проведения различных форм частотного и корреляционного криптоанализа. Во втором же все проще, так как повторное наложение ИЛИ на новый исходный блок p2 дает открытый текст, с помощью формулы:

$$(p1 \text{ XOR} g) \text{XOR} (p2 \text{ XOR} g) = p1 \text{ XOR} p2.$$

Что бы избежать таких неловких моментов, шифр гаммирования должен упираться на два критерия:

- Гамма, должна иметь очень хорошие статистические характеристики.
- Период гаммы должен превышать длину самого большого сообщения, которые должны быть переданы и зашифрованы с помощью одного ключа.

В каналах где возможно модификация данных при передачи, такие шифры беззащитны. Сделав модификацию принимающая сторона не сможет расшифровать данные. Поэтому в канала такого типа нужно реализации подтверждения целостности сообщения. А это уже реальные угрозы информационной безопасности.

Синхронные поточные шифры

Синхронные поточные шифры (СПШ) — шифры, где поток ключей генерируется независимо от шифротекста и открытого текста. В процессе шифрования генератор потока ключей дает биты потока ключей, которые идентичны битам потока ключей при процессе дешифрования. При потере любого знака шифротекста нарушается синхронизация между двумя генераторами и возможность расшифровать оставшиеся куски текста.

Синхронизация обычно реализуется через специальные вставные маркеры. При потере знака рассинхроность текста будет незначительно, а именно до следующего маркера. После него синхронность опять настроена.

Плюсы синхронного шифра:

- Отсутствие эффекта распространения ошибок (только искаженные биты будут неправильно расшифрованны)
- Страхует от любых вставок и удаление шифротекста, так как поток будет рассинхронизирован и будут выявлены

Минусы синхронного шифра:

- Уязвим к изменению отдельных бит шифрованного текста.
- Если известен открытый текст, то можно изменить биты таким образом, что бы расширование было подделано

Структурная схема алгоритма шифрования из лабораторной работы

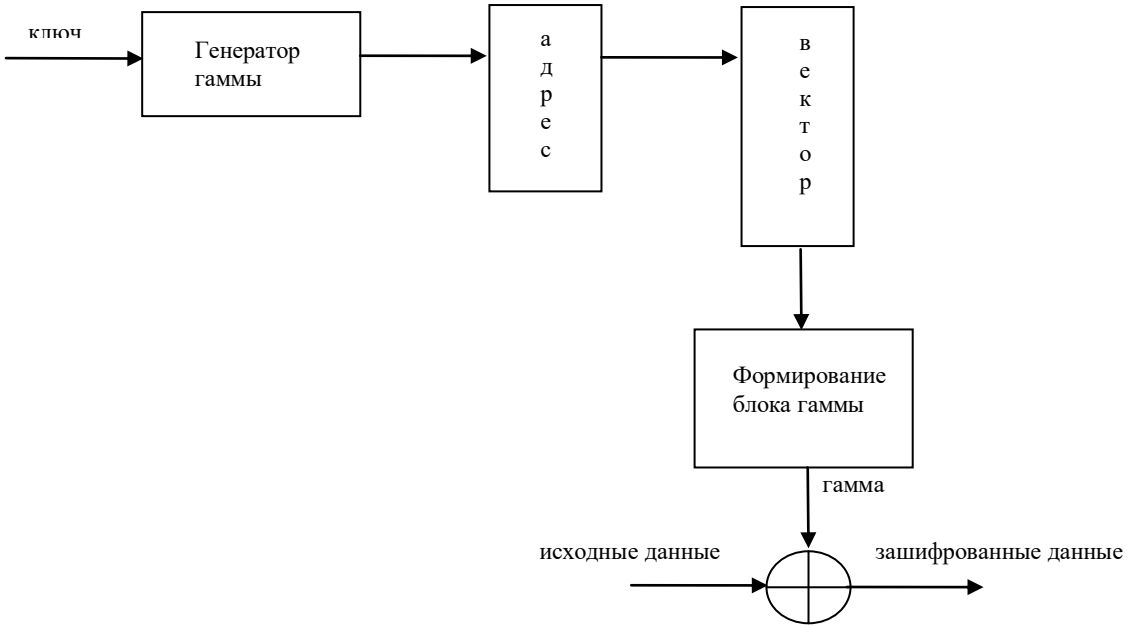

рисунок 3 - Структурная схема алгоритма шифрования

Описание хода выполнения работы

Для начала опишем алгоритм из лабораторной работы с полученными результатами, а после опишем модифицированный алгоритм шифрования и приведет тесты на решетчатость и автокорреляци.

В качестве генератора контрольной гаммы использовался линейный конгуэнтный генератор Лемера с параметрами:

- m = 2^31-1;
- a = 7^5;
- c = 0;

Алгоритм формирования случайного числа выглядит следующим образом :

$X_{n+1} = (aX_n + c) \bmod m$, где m – модуль, a – множитель, c – приращение, mod – операция взятия остатка от деления.

На вход генератора поступает ключ, который мы сами задаем, он является секретной информацией.

Я в своем исследований я использовал 4 битный адрес, поэтому из полученного 30 разрядного числа с генератора достаю 4 случайных бита, которые сформируют адрес, на который сошлется вектор и извлечет бит из данного положения.

Так как адрес 4 битный, то вектор состоит из 2^4=16 значений. Важно чтобы распределение 0 и 1 в векторе было равновероятным, иначе это отразится на качестве шифрования.

Полученный бит из вектора формирует гамму нужной длинный, в данном исследовании длинна гаммы равна 16, следовательно чтобы сформировать один блок гаммы, необходимо 16 тактов работы программы.

После того как блок гаммы сформирован необходимо выполнить операцию сложения по модулю два между исходными данными и блоком гаммы, и на выходе получим зашифрованных блок. Алгоритм будет продолжать свою работу, пока не зашифруются все данные.

Дешифрование данных происходит аналогичным образом , где складывается по модулю два зашифрованные данные с блоком гаммы. Процесс шифрования и дешифрования с блоком гаммы схематично изображен на рисунке 4.

рисунок 4 - процесс шифрования и дешифрования с блоком гаммы

Модификация алгоритма

Для того чтобы внести непредсказуемость в наш алгоритм шифрования были реализованы следующие решения.

Каждые n тактов, в зависимости от объема шифрованных данных изменялся алгоритм генерации случайных чисел в генераторе гаммы. В зависимости от того какой бит был на выходе у вектора и старший разряд в гамме 0 или 1 изменялся алгоритм следующим образом:

- Если 1, то $X_{n-1} = X^3_{n-1}$;
- если 0, то $X_{n-1} = X^3_{n-100}$;

Так же через n тактов изменялся вектор, производился циклический сдвиг либо влево либо вправо, и на случайной число элементов.

Полученные результаты

Исходное изображение

рисунок 5 - исходное изображение

Зашифрованное изображение

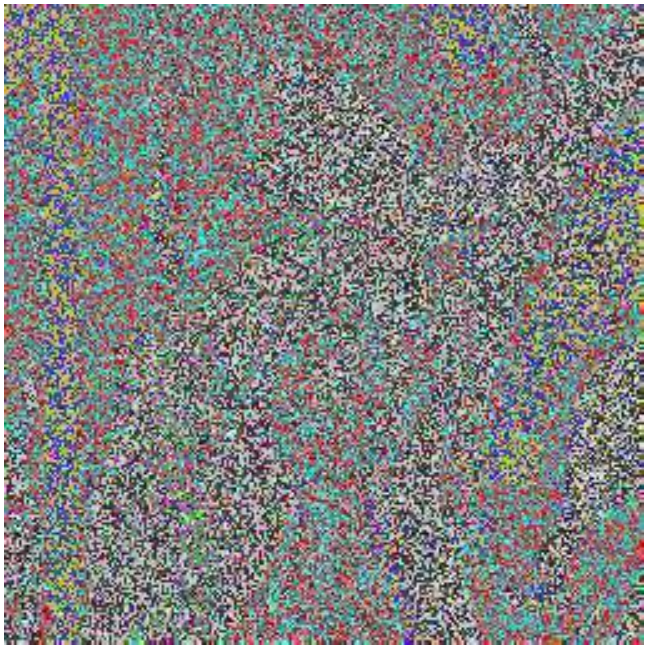

рисунок 6 - зашифрованное изображение

дешифрованное

рисунок 7 - дешифрованное изображение

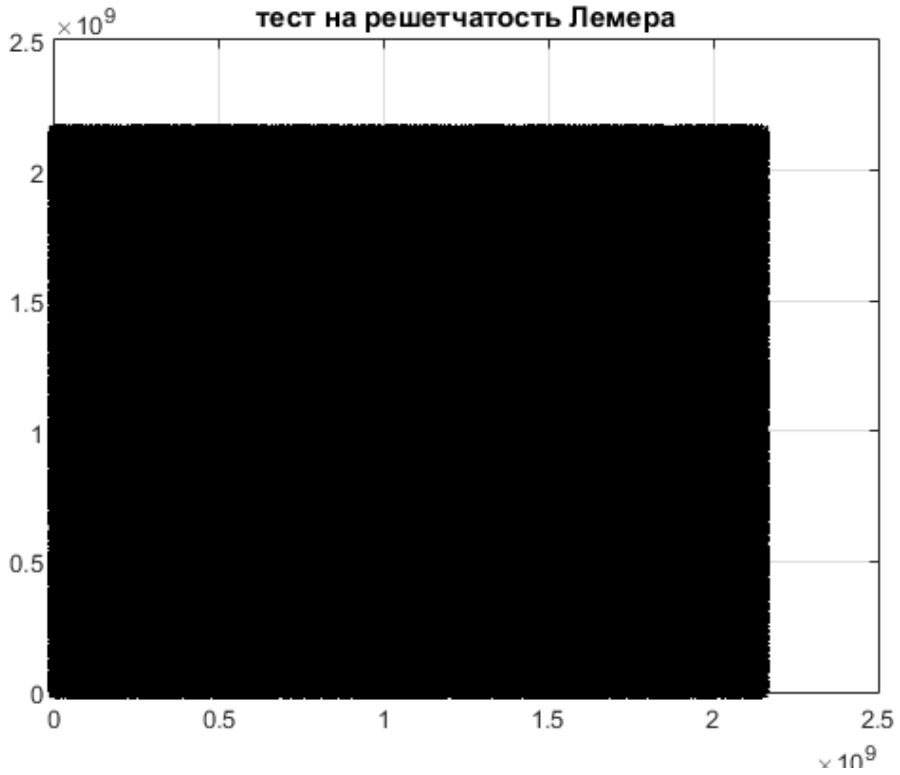

рисунок 8 - тест на решетчатость для генератора Лемера

рисунок 9 - автокорреляционная функция для алгоритма Лемера

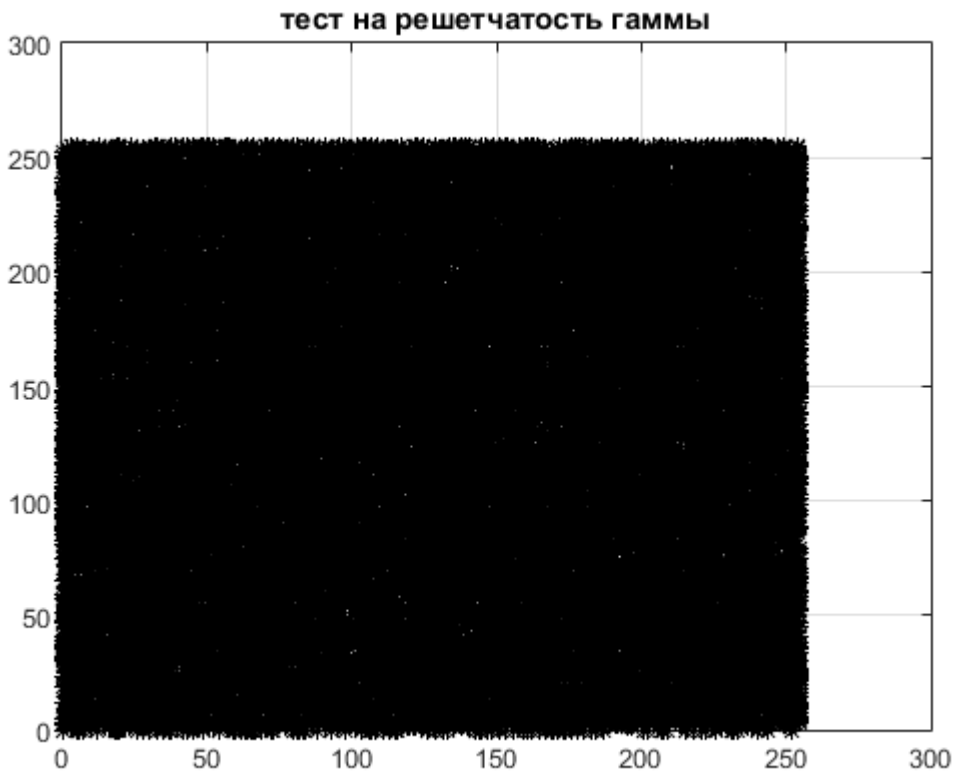

рисунок 10 - тест на решетчатость для гаммы

рисунок 11 - автокорреляционная функция для гаммы

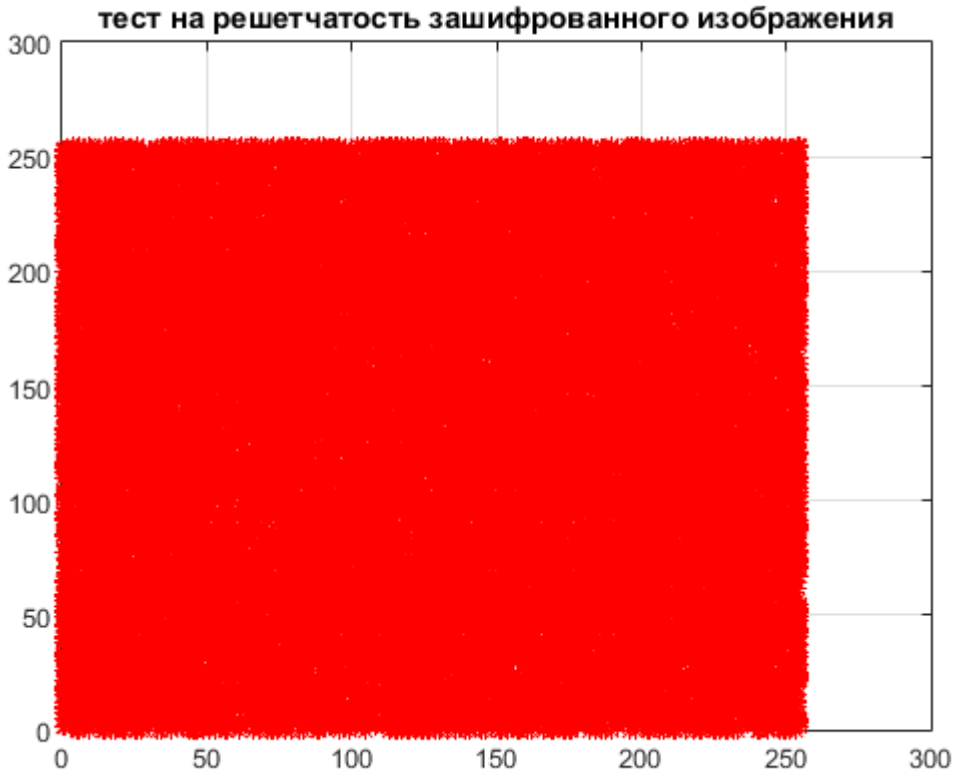

рисунок 12 - тест на решетчатость для шума изображения

Если в векторе будет не равновероятное распределение 0 и 1 то получим следующие результаты:

рисунок 13 - зашифрованное изображения для случая, когда 0 больше 1

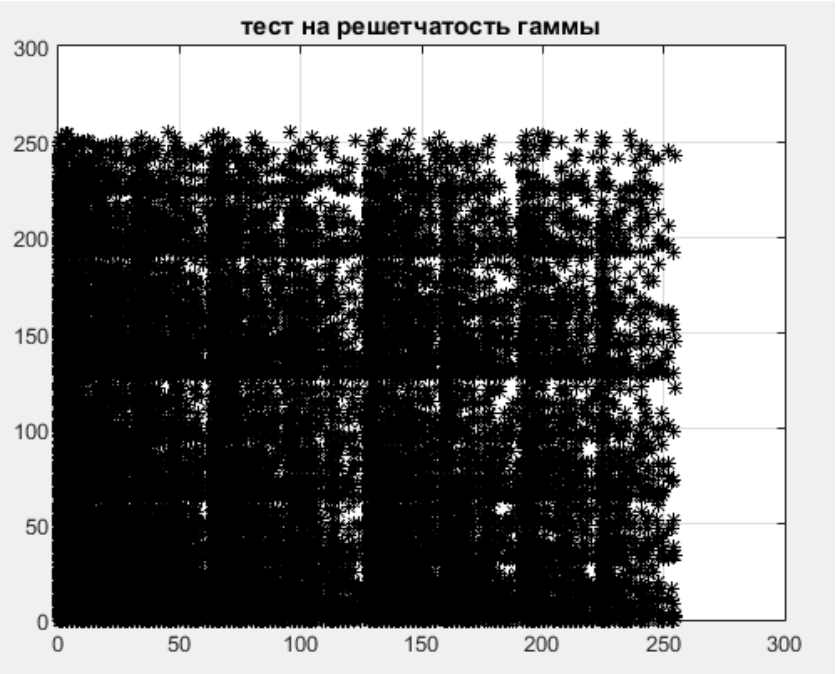

рисунок 14 - тест на решетчатость для гаммы для случая, когда 0 больше 1

рисунок 15 - зашифрованное изображения для случая, когда 1 больше 0

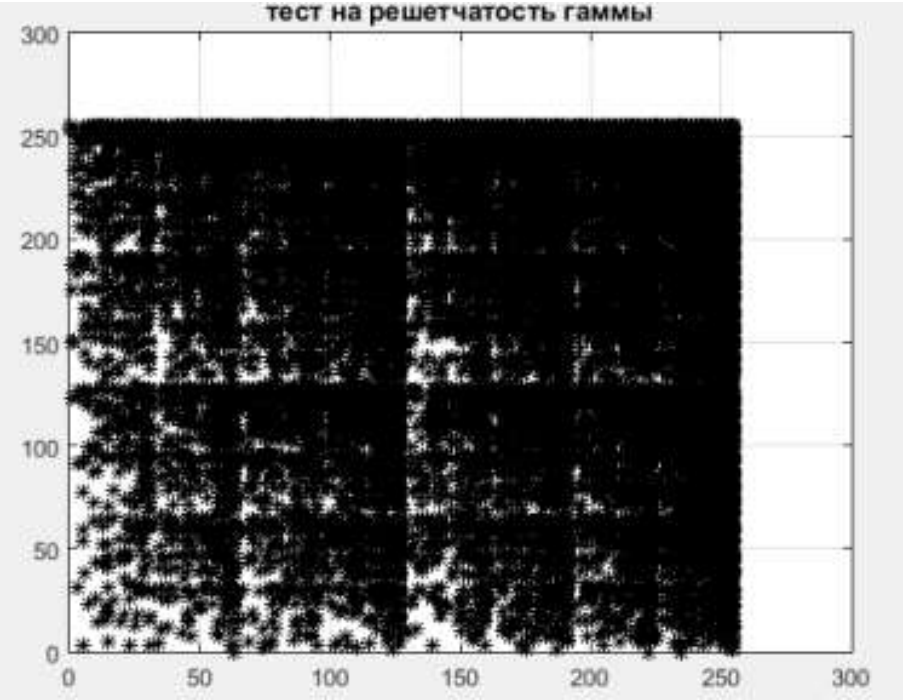

рисунок 16 - тест на решетчатость для гаммы для случая, когда 1 больше 0

Оценка сложности взлома алгоритма

Параметры, которые можно вычислить и дешифровать наше сообщение, это числа генератора и возможное размещение вектора, следовательно чтобы оценить сложность, перемножим всевозможные варианты этих двух параметров.

Получаем:

$$(2^{31}-1)*2*C_{16}^{8} = 2147483640 \ *2*12870 = 55\,276\,228\,893\,600.$$

Выводы

В ходе выполнения лабораторной работы были реализован потоковый алгоритм шифрование, который носит название синхронное гаммирование.

Структура алгоритма в виде схемы изображена на рисунке 3. На вход генератора псвевдослучайных чисел подавалось число (ключ), который формировал выходное зерно генератора. Из полученного числа брались случайные n бит, которые формировали адрес, на который ссылался вектор для выделения бита 0 или 1 для построения гаммы.

Вектор который формирует гамму должен быть заполнен равновероятно 0 и 1, так как если это будет не так, шифрование пройдет плохо. Преобладание 0 в векторе при шифровании выдаст на выходе контуры изображения, так как это показано на рисунке 13. Преобладание 1 в векторе при шифровании на выходе получим просто инвертированные цвета изображения, так как это показано на рисунке 15.

Также было построен тест на решетчатость для этих двух случаев, на которых видно скопление в сторону 0 или 255 у зашифрованного изображения, как это показано на рисунках 14 и 16.

Для шифрования использовалось изображение в формате BMP-24, в нем на каждую компоненту отводится 8 бит. На выходе получаем шифрованное изображение, и если вектор состоит из 0 и 1 равновероятно, то на изображение накладывается такой шум, где при зрительном оценивании обнаружить , какое изображение шифровалось невозможно, так как это представлено на рисунке 6.

Если противник получит ключ генератора и вектор , то он сможет расшифровать наше изображение. Для того чтобы сделать эту задачу было решить сложнее, внесем дополнительную непредсказуемость в алгоритм шифрования.

Каждые n тактов будет изменять наш вектор, путем сдвига влево или вправо, и на случайное число позиций. Также будет изменять ключ генератора через n тактов, в зависимости от старшего бита полученной гаммы, по правилу если был 0, то предыдущее число возводим в 3 степень, а если была 1, то возводим в 3 степень число, которое было 100 тактов назад.

Так как мы меняем параметры генератора, необходимо проверить, что числа которые поступают с его выхода все еще псевдослучайны, для этого построим тест на решетчатость для него, и видим что случайность сохранилась, как на рисунке 8.

Так же необходимо с таким подходом рассмотреть генерацию гаммы, для этого построили тест на решетчатость, который так же дал хороший результат распределения чисел, как показано на рисунке 10.

Была оценена сложность алгоритма численно, число которое получилось будет не быстро перебрать брать методом грубой силы. Численно оно равно : 55 276 228 893 600 .

ПРИЛОЖЕНИЕ

Листинг программ реализующих данный алгоритм

```
close all;
clear all;
clc;

%% генертаор Лемера
H=256;
W=256;
N=H*W*8;
x=zeros(1,N-1);
x1=12345678;
x=[x1,x];
c=0;
b=7^5;
p=(2^31)-1;
F=[0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1]; %поиск в векторе
%F= randint(1,16);%начальная растановка вектор
gamma=zeros(1,N);
for i=2:1:N+1

    if mod(i,1000)==0
        if gamma(1,i-1)==0
        n=x(1,i-1)^3;
        else
        n=x(1,i-100)^3;
        end
             x(1,i)=mod((n*b+c),p);
    else
    x(1,i)=mod((x(1,i-1)*b+c),p);
    end
%% гаммирование
%переводим в двоичный
A=de2bi(x(1,i));
%формируем адрес
adres=A(1,1:4);
%вектор логической функции
%
if mod(i,1000)==0
 F = circshift(F',3);%меняем вектор
 F=F';
end

position_in_vector=bi2de(adres)+1;

gamma(1,i)=F(position_in_vector);
end
```

```
%% тест на решетчатость
figure(1);
ne_chet=x(1,1:2:end-1);
chet=x(1,2:2:end);

plot(chet,ne_chet,'black*');
 title('тест на решетчатость Лемера');
 grid on

%% автокорреляция
 G  = autocorr(x,10000);
 figure(2)
 plot(G);
 title('автокорееляционная функция Лемера');
 grid on
r=1;
MM=H*W;
gamma1=zeros(MM,8);
for i=1:1:MM
    for j=1:1:8
        gamma1(i,j)=gamma(1,r);

        r=r+1;
    end
end
%гаммаааааа
gamma2=bi2de(gamma1);
cor_gamma=autocorr(gamma2,10000);
figure(3)
 plot(cor_gamma);
 title('автокорееляционная функция гаммы');
 grid on

 %% тест на решетчатость gamma
figure(4);
ne_chet_gamma=gamma2(1:2:end,1);
chet_gamma=gamma2(2:2:end,1);

plot(chet_gamma,ne_chet_gamma,'black*');
 title('тест на решетчатость гаммы');
 grid on

%считали картинку
%Image_color=imread('C:\Users\Виктор\Desktop\1_курс_магистратура
\окатов\Новая папка\2лаба\lena512.bmp');%источник цвета
Image_color=imread('C:\Users\Виктор\Desktop\1_курс_магистратура\
окатов\Новая папка\2лаба\передел\le.bmp');
figure(5);
imshow(Image_color);
title('Исходное изображение');

%записали каждую компоненту
Image_RED=Image_color(:,:,1);
```

```
Image_g=Image_color(:,:,2);
Image_b=Image_color(:,:,3);
%шифрование
[ enc_im,vostanov_r,vostanov_g,vostanov_b ]  = gammirovanie(
Image_RED, Image_g,Image_b,gamma1,H,W);
figure(6);
imshow(uint8(enc_im));
title('Зашифрованное изображение');
%% тест картинки зашифров
encr_im_1=zeros(1,H*W);
encr_im_2=zeros(1,H*W);
encr_im_3=zeros(1,H*W);
g=1;
for i=1:1:H
    for j=1:1:W
            encr_im_1(1,g)=enc_im(i,j,1);
            encr_im_2(1,g)=enc_im(i,j,2);
            encr_im_3(1,g)=enc_im(i,j,3);

    g=g+1;
    end
end
figure(8);
ne_chet_encr_im_1=encr_im_1(1,1:2:end-1);
chet_encr_im_1=encr_im_1(1,2:2:end);
plot(chet_encr_im_1,ne_chet_encr_im_1,'r*');
 title('тест на решетчатость зашифрованного изображения');
 grid on
 figure(9);
ne_chet_encr_im_2=encr_im_2(1,1:2:end-1);
chet_encr_im_2=encr_im_2(1,2:2:end);
plot(chet_encr_im_2,ne_chet_encr_im_2,'g*');
 title('тест на решетчатость зашифрованного изображения');
 grid on
 figure(10);
ne_chet_encr_im_3=encr_im_3(1,1:2:end-1);
chet_encr_im_3=encr_im_3(1,2:2:end);
plot(chet_encr_im_3,ne_chet_encr_im_3,'b*');
 title('тест на решетчатость зашифрованного изображения');
 grid on
%% дешифрование
dec_im  =degammirovanie(
vostanov_r,vostanov_g,vostanov_b,gamma1,H,W );

figure(7);
imshow(uint8(dec_im));
title('дешифрованное');
```

Функция шифрования

```
function [ enc_im,vostanov_rr,vostanov_gg,vostanov_bb ] =
gammirovanie( Image_RED, Image_g,Image_b,gamma,H,W)
```

```
%перевод из 10 в 2 все интенсивности
bin_im_red=de2bi(Image_RED);
bin_im_g=de2bi(Image_g);
bin_im_b=de2bi(Image_b);
%шифрование картинки и гаммы
Im_enc_red=xor(bin_im_red(:,:),gamma(:,:));
Im_enc_g=xor(bin_im_g(:,:),gamma(:,:));
Im_enc_b=xor(bin_im_b(:,:),gamma(:,:));
%восстановение из 2 в 10
vostanov_rr=zeros(H,W);
vostanov_gg=zeros(H,W);
vostanov_bb=zeros(H,W);
l=1;
for i=1:1:W
    for j=1:1:H
        vostanov_rr(j,i)=bi2de(Im_enc_red(l,:));
        vostanov_gg(j,i)=bi2de(Im_enc_g(l,:));
        vostanov_bb(j,i)=bi2de(Im_enc_b(l,:));
        l=l+1;
    end
end
enc_im(:,:,1)=vostanov_rr;
enc_im(:,:,2)=vostanov_gg;
enc_im(:,:,3)=vostanov_bb;
end
```

Функция дешифрования

```
function [ dec_im ] =degammirovanie(
vostanov_r,vostanov_g,vostanov_b,gamma,H,W )
bin_im_red_dec=de2bi(vostanov_r);
bin_im_g_dec=de2bi(vostanov_g);
bin_im_b_dec=de2bi(vostanov_b);
Im_dec_red=xor(bin_im_red_dec(:,:),gamma(:,:));
Im_dec_g=xor(bin_im_g_dec(:,:),gamma(:,:));
Im_dec_b=xor(bin_im_b_dec(:,:),gamma(:,:));
vostanov_r_dec=zeros(H,W);
vostanov_g_dec=zeros(H,W);
vostanov_b_dec=zeros(H,W);
l=1;
for i=1:1:W
    for j=1:1:H
        vostanov_r_dec(j,i)=bi2de(Im_dec_red(l,:));
        vostanov_g_dec(j,i)=bi2de(Im_dec_g(l,:));
        vostanov_b_dec(j,i)=bi2de(Im_dec_b(l,:));
        l=l+1;
    end
end
dec_im(:,:,1)=vostanov_r_dec;
dec_im(:,:,2)=vostanov_g_dec;
dec_im(:,:,3)=vostanov_b_dec;

end
```