ГУАП

Институт киберфизических систем
Кафедра ИБ

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

доц., канд. техн. наук		А.В. Афанасьева
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ

Внедрение информации в область преобразования

по курсу: Технологии стеганографии в системах инфокоммуникаций

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 3931		В.С. Кулешов
	подпись, дата	инициалы, фамилия

Санкт-Петербург, 2022 г.

1 Цель работы

Целью данной лабораторной работы является разработка стегосистемы на основе внедрения информации в изображение в формате *.bmp* методом Cox. С вставкой по формуле:

$$c_i^{'} = c_i e^{\alpha s_i}$$

2 Ход выполнения работы

Алгоритм закрытия информации:

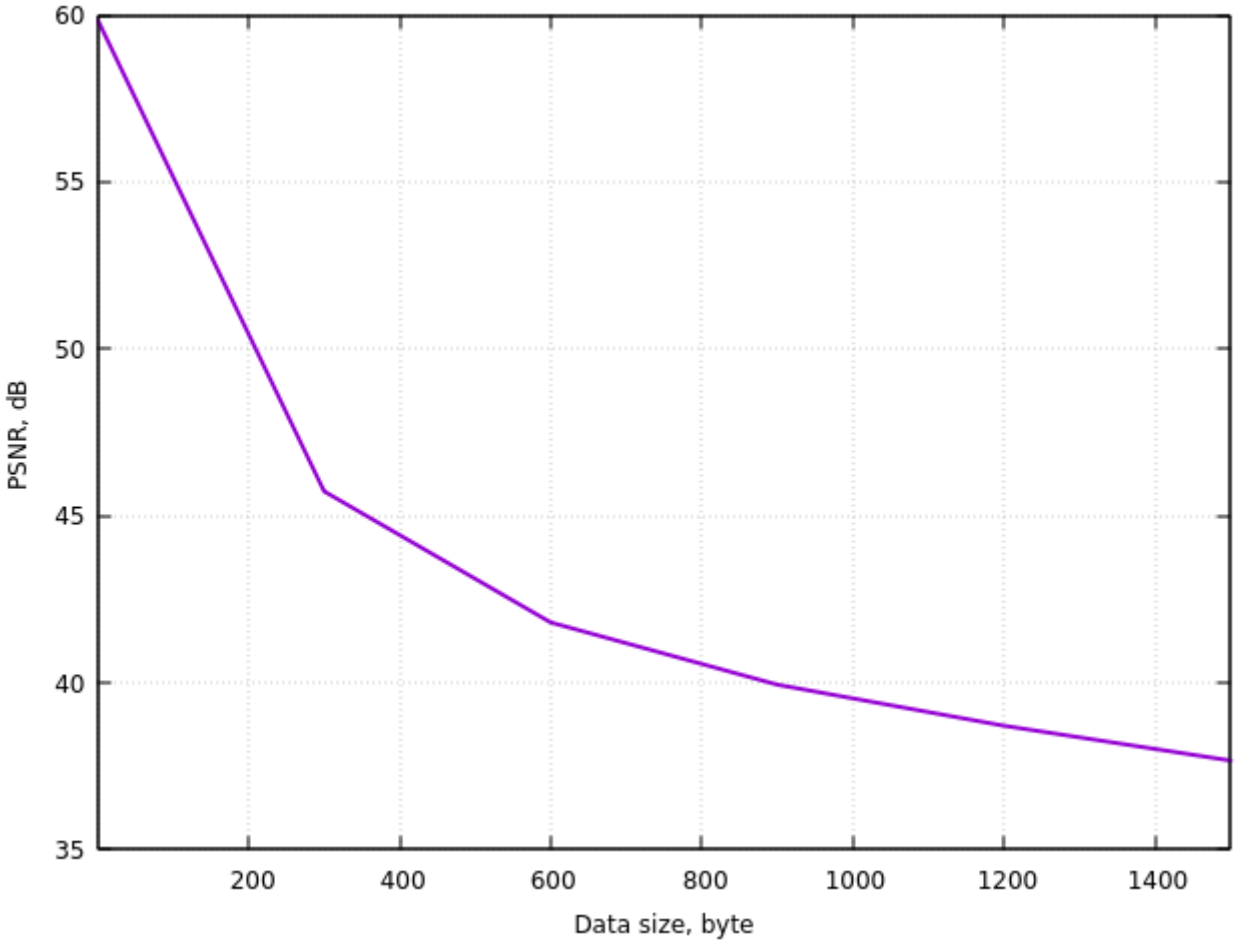
1. Данные представляются в виде бинарной последовательности;
2. Изображение-контейнер разбивается на блоки 8x8;
3. Последовательно к каждому блоку применяется дискретно-косинусное преобразование (ДКП). Наибольший коэффициент полученного блока изменяется по формуле из условия;
4. К измененным коэффициентам применяется обратное ДКП.

Алгоритм раскрытия информации:

1. Изображение-контейнер разбивается на блоки 8x8;
2. Поочередно к каждому блоку применяется ДКП, а так же к соответствующему блоку открытого контейнера;
3. На основе сравнения наибольших коэффициентов выносится решение о значении закрытого бита информации.

Санкт-Петербург, 2022 г.

2. 1 Влияние на качество изображения



3 Вывод

В результате выполнения лабораторной работы была реализована стегосистема в соответствии с предъявленными требованиями.

Санкт-Петербург, 2022 г.

4 Листинг кода

```
bool
BMPContainerExtended::CellExtended::get()
const
{
        if (!is_origin_loaded)
            throw std::runtime_error("Can't execute CellExtended::get function without loaded original file.");

        std::array<std::array<double, BLOCK_SIZE>, BLOCK_SIZE> modified{}, origin{};
        for (int i = 0; i < BLOCK_SIZE; ++i)
            for (int j = 0; j < BLOCK_SIZE; ++j)
                switch (color_) {
                case 0:
                    modified[i][j] = pic.GetPixel(y_ + j, x_ + i).Red;
                    origin[i][j] = pic_orig.GetPixel(y_ + j, x_ + i).Red;
                    break;

                case 1:
                    modified[i][j] = pic.GetPixel(y_ + j, x_ + i).Green;
                    origin[i][j] = pic_orig.GetPixel(y_ + j, x_ + i).Green;
                    break;

                case 2:
                    modified[i][j] = pic.GetPixel(y_ + j, x_ + i).Blue;
                    origin[i][j] = pic_orig.GetPixel(y_ + j, x_ + i).Blue;
                }

        auto transformed = dct(modified);
        auto [i_max, j_max] = get_max_idx(transformed);

        return (modified[i_max][j_max] >= origin[i_max][j_max]);
}
```

```
void
BMPContainerExtended::CellExtended::set(bool v)
{
        auto func = [this, v] () {
            std::array<std::array<double, BLOCK_SIZE>, BLOCK_SIZE> origin{};
            for (int i = 0; i < BLOCK_SIZE; ++i)
                for (int j = 0; j < BLOCK_SIZE; ++j)
                    switch (color_) {
                    case 0:
                        origin[i][j] = pic.GetPixel(y_ + j, x_ + i).Red;
                        break;

                    case 1:
                        origin[i][j] = pic.GetPixel(y_ + j, x_ + i).Green;
                        break;

                    case 2:
                        origin[i][j] = pic.GetPixel(y_ + j, x_ + i).Blue;
                    }

            auto transformed = dct(origin);
            auto [i_max, j_max] = get_max_idx(transformed);
            transformed[i_max][j_max] *= E_ALPHA[v];

            auto result = idct(transformed);
            for (int i = 0; i < BLOCK_SIZE; ++i)
                for (int j = 0; j < BLOCK_SIZE; ++j)
                {
                    std::lock_guard<std::mutex> lk{pic_mtx};
                    auto pxl = pic.GetPixel(y_ + j, x_ + i);

                    switch (color_) {
                    case 0:
                        pxl.Red = result[i][j];
                        break;

                    case 1:
                        pxl.Green = result[i][j];
                        break;

                    case 2:
                        pxl.Blue = result[i][j];
                    };

                    pic.SetPixel(y_ + j,
                                 x_ + i,
                                 pxl);
                }
        };

        futures.emplace_back(std::async(std::launch::async, func));
}
```