Учреждение образования

«Белорусский государственный технологический университет»

**Кафедра «Информационных систем и технологий»**

**Лабораторная работа №14**

Исследование стеганографического метода на основе преобразования наименее значащих бит

Выполнил:

Студент 3 курса, 10 группы

Баранчук Владислав

Минск, 2021 г.

**Цель:** изучение стеганографического метода осаждения/ извлечения тайной информации с использованием электронного файла-контейнера на основе преобразования наименее значащих бит (НЗБ), приобретение практических навыков программной реализации данного метода

**Стеганографическая система** – совокупность средств и методов, которые используются для формирования скрытого канала передачи (или хранения) информации. При этом скрытый канал организуется на базе и внутри открытого канала с использованием особенностей восприятия информации. «Скрытость» канала передачи тайной информации отличает стеганографии от криптографии: в первом случае тайной является сам факт наличия канала (передачи информации). Абстрактно стеганографическая система обычно определяется, как некоторое множество отображений одного пространства (множества возможных сообщений, М) в другое пространство (множество возможных стеганосообщений, S, и наоборот.

Основные компоненты стеганосистемы:

* *контейнер, С*
* *тайное сообщение, М;*
* *ключи или ключевая информация, K;*
* *контейнер с осажденным сообщением или стеганоконтейнер, S;*
* *отправителя и получателя.*

Стеганографической системой ∑ будем называть совокупность сообщений M, контейнеров C, ключей K, стеганосообщений (заполненных контейнеров) S и преобразований (прямого F и обратного F -1), которые их связывают:



Метод НЗБ основывается на ограниченных способностях зрения или слуха человека, вследствие чего людям тяжело различать незначительные вариации цвета или звука. Рассмотрим это на примере 24-битного растрового RGB-изображения. Как известно, каждая точка кодируется 3-мя байтами. Каждый байт определяет интенсивность красного (Red), зеленого (Green) и синего (Blue) цветов. Совокупность интенсивностей цвета в каждом из 3-х каналов определяет оттенок пикселя.

**Практическая часть**

Листинг приложения, реализующего метод НЗБ, написанного на JAVASCRIPT

var

    canvas = document.getElementById( 'canvas' ),

    secret = document.getElementById( 'secret' ),

    coverAfter = document.getElementById( 'coverAfter' ),

    ctx = canvas.getContext( '2d' ),

    ctxSecret = secret.getContext( '2d' ),

    ctxCoverAfter = coverAfter.getContext( '2d' ),

    myFile = document.getElementById( 'myFile' ),

    loadFile = document.getElementById( 'loadFile' ),

    view,

    clampedArray,

    index = 0;

    // прослушиваем изменение события

    // это событие, будет хранить Int8Array внутри переменной представления

myFile.addEventListener( 'change', function ( e ) {

    var file = e.target.files[ 0 ];

    var fr = new FileReader();

    fr.addEventListener( "load", loadEvent );

    function loadEvent ( evt ) {

        if ( evt.target.readyState == FileReader.DONE ) {

              // возвращенный объект arraybuffer

            var arrayBuffer = evt.target.result;

              // назначаем массив буферов со знаком int8array вместо обычного массива

             // 1 байт на индекс и сохраняем только до 256 значений

            view = new Uint8Array( arrayBuffer );

        }

    }

    // читаем как массив буферов

    fr.readAsArrayBuffer( file );

});

hideData.addEventListener( 'click', function (e ) {

    var

        cover = document.getElementById( 'cover' ),

        file = cover.files[ 0 ],

        fr = new FileReader();

    fr.addEventListener( "load", loadEvent );

    fr.addEventListener( "loadend", loadEndEvent );

    function loadEvent ( e ) {

        console.info( 'load just start..wait for finish' );

    }

    function loadEndEvent ( e ) {

        console.info( 'finish' );

        var img = new Image();

        img.src = e.target.result;

       // ждем окончания загрузки изображения

       // затем выводим изображение на экран

        img.onload = function () {

            // вывод картинки

            ctx.drawImage( img, 0, 0 );

           // get imageData - состоит из значения rgba для каждого индекса массива

            clampedArray = ctx.getImageData( 0, 0, canvas.height, canvas.width );

            console.log( clampedArray );

            console.log( view );

            // начинаем читать и заменять биты

            readByte( view );

            //console.log(clampedArray);

            // рисуем холст, используя вместо этого данные изображения или объект img

            ctxSecret.putImageData( clampedArray, 0, 0 );

        }

    }

    // читаем как url данных (base64)

    fr.readAsDataURL( file );

});

loadFile.addEventListener( 'change', function ( e ) {

    var file = e.target.files[ 0 ];

    var fr = new FileReader();

    fr.addEventListener( "loadend", loadEndEvent );

    function loadEndEvent ( e ) {

        var img = new Image();

        img.src = e.target.result;

       // ждем окончания загрузки изображения

       // затем рисуем изображение на холсте

        img.onload = function () {

            ctxCoverAfter.drawImage( img, 0, 0 );

            // возвращенный объект arraybuffer

            // var arrayBuffer = evt.target.result;

            // присваиваем массиву буферов без знака int8array вместо обычного массива

            // 1 байт на индекс и сохраняем только до 256 значений

            // var loadView = new Uint8Array (arrayBuffer);

            var loadView = ctxCoverAfter.getImageData( 0, 0, coverAfter.height, coverAfter.width );

            console.log( loadView )

            var totalLength = 0;

            var lastIndex;

            // перебираем бит пикселя

            // суммируем всю длину (длину секретных данных)

            for ( var b = 0, viewLength = loadView.data.length; b < viewLength; b++ ) {

                // получаем длину только для сопоставленного индекса

                if (loadView.data[ b ] == 255) {

                    totalLength += loadView.data[ b ];

                    if (loadView.data[ b + 1 ] < 255) {

                        totalLength += loadView.data[ b + 1 ];

                        lastIndex = b + 1;

                        break;

                    }

                } else {

                    totalLength += loadView.data[ b ];

                    lastIndex = b;

                    break;

                }

            }

            console.info( 'Total length :' + totalLength + ', Last Index : ' + lastIndex )

                // получаем первый индекс - длина секрета

            var secretLength = totalLength;

            // создаем экземпляр беззнакового массива (8 бит)

            // делится на 4 как один код символа, равный 8 битам

            var newUint8Array = new Uint8Array( totalLength / 4 );

            var j = 0;

            // начинаем извлекать биты из пикселя

            for ( var i = ( lastIndex + 1 ); i < secretLength; i = i + 4 ) {

                 // нам нужны только первые 2 бита от каждого байта

                 // поскольку эти 2 бита содержат бит наших секретных данных

                 // сначала очищаем неиспользованный бит с помощью mask (3) == 0000 0011

                 // затем сдвиг влево для каждого бита (упорядочение)

                 // оставаясь на своем месте

                var aShift = ( loadView.data[ i ] & 3 );

                var bShift = ( loadView.data[ i + 1 ] & 3 ) << 2;

                var cShift = ( loadView.data[ i + 2 ] & 3 ) << 4;

                var dShift = ( loadView.data[ i + 3 ] & 3 ) << 6;

                // объединить / объединить все сдвинутые биты, чтобы сформировать байт (8 бит)

                var result = ( ( ( aShift | bShift) | cShift ) | dShift );

                // сохранить результат (один байт) в целое число без знака

                newUint8Array[ j ] = result;

                j++;

            }

            console.log( newUint8Array )

            // декодировать коллекцию целых чисел без знака в набор символов ASCII

            var result = decodeUtf8( newUint8Array );

            console.log( result )

            // принудительно загружать результаты в файлы .txt

            saveByteArray( result.split(''), 'STGSAV.txt' );

        }

    }

    // читаем как dataUrl (base64)

    fr.readAsDataURL( file );

});

//чтение секретного бита для кода каждого набора символов

function readByte( secret ) {

    for ( var i = 0, length = secret.length; i < length; i++ ) {

        if ( i == 0 ) {

           // на первом бите, сохраняем длину секретных данных

           // должен умножиться на 4, так как код одного символа содержит

           // 8 бит, таким образом, эти 8 бит делятся на 2. каждые 2 бита должны заменить

           // младший значащий бит байта пикселя

            var secretLength = length \* 4;

            console.info( 'Secret Length(' + length + 'x4) : ' + secretLength )

            // поскольку наш imageData является типизированным массивом (Uint8ClampedArray)

            // он может хранить только значение не более 256 (8 бит или 1 байт)

            if ( secretLength > 255 ) {

                // проверяем, сколько раз нам нужен индекс imageData

                // для хранения длины вашего секрета

                var division = secretLength / 255;

                // целое число

                if ( division % 1 === 0 ) {

                    for ( var k = 0; k < division; k++ ) {

                        clampedArray.data[ k ] = 255;

                        index++;

                    }

                }

                // вещественное число

                else {

                    var firstPortion = division.toString().split(".")[ 0 ];

                    var secondPortion = division.toString().split(".")[ 1 ];

                    for ( var k = 0; k < firstPortion; k++ ) {

                        clampedArray.data[ k ] = 255;

                        index++;

                    }

                    var numberLeft = Math.round( ( division - firstPortion ) \* 255 );

                    console.info( 'numberLeft : ' + numberLeft )

                    clampedArray.data[ k ] = numberLeft;

                    index++;

                }

            } else {

                clampedArray.data[ 0 ] = secretLength;

                index++;

            }

            console.log( 'sss : ' + clampedArray.data[ 0 ] )

        }

        var asciiCode = secret[ i ];

        // используем маскировку, чтобы очистить бит, и взять бит, который мы только хотим

        // Взять только первые 2 бита, например: 0111 0011 => 0000 0011

        var first2bit = ( asciiCode & 0x03 ); // 0x03 = 3

        // Взять только первые 4 бита (2 бита в конце), например: 0111 0011 => 0000 0000

        var first4bitMiddle = ( asciiCode & 0x0C ) >> 2;

       // Взять только первые 6 бит (2 бита в конце), например: 0111 0011 => 0011 0000

        var first6bitMiddle = ( asciiCode & 0x30 ) >> 4;

       // Взять только первые 8 бит (2 бита в конце), например: 0111 0011 => 0100 0000

        var first8bitMiddle = ( asciiCode & 0xC0 ) >> 6;

        //console.log(i + ' : ' + first2bit);

        //console.log(i + ' : ' + first4bitMiddle);

        //console.log(i + ' : ' + first6bitMiddle);

        //console.log(i + ' : ' + first8bitMiddle);

        // начинаем заменять бит нашего секрета на LSB

        replaceByte( first2bit );

        replaceByte( first4bitMiddle );

        replaceByte( first6bitMiddle );

        replaceByte( first8bitMiddle );

    }

}

//заменить биты для каждого байта imageData

function replaceByte ( bits ) {

    // очистить первые два бита (LSB), используя &

    // и замена секретным битом

    clampedArray.data[ index ] = ( clampedArray.data[ index ] & 0xFC ) | bits;

    index++;

}

 // сохранить / принудительно загрузить данные

var saveByteArray = (function() {

    var a = document.createElement("a");

    document.body.appendChild(a);

    a.style = "display: none";

    return function(data, name) {

        var blob = new Blob(data, {

                type: "octet/stream"

            }),

            url = window.URL.createObjectURL(blob);

        a.href = url;

        a.download = name;

        a.click();

        window.URL.revokeObjectURL(url);

    };

}());

// декодировать код символа в символ

function decodeUtf8(arrayBuffer) {

    var result = "";

    var i = 0;

    var c = 0;

    var c1 = 0;

    var c2 = 0;

    var data = new Uint8Array(arrayBuffer);

// если у нас есть спецификация, пропустите

    if (data.length >= 3 && data[0] === 0xef && data[1] === 0xBB && data[2] === 0xBF) {

        i = 3;

    }

    while (i < data.length) {

        c = data[i];

        if (c < 128) {

            result += String.fromCharCode(c);

            i++;

        } else if (c > 191 && c < 224) {

            if (i + 1 >= data.length) {

                throw "UTF-8 Decode failed. Two byte character was truncated.";

            }

            c2 = data[i + 1];

            result += String.fromCharCode(((c & 31) << 6) | (c2 & 63));

            i += 2;

        } else {

            if (i + 2 >= data.length) {

                throw "UTF-8 Decode failed. Multi byte character was truncated.";

            }

            c2 = data[i + 1];

            c3 = data[i + 2];

            result += String.fromCharCode(((c & 15) << 12) | ((c2 & 63) << 6) | (c3 & 63));

            i += 3;

        }

    }

    return result;

}

Программа принимает в себя исходный текстовый файл с информацией, исходное изображение, в которое будет интегрироваться с помощью метода НЗБ секретная информация. Далее выдает изображение со скрытой информацией, которое никак не различимо человеческому взгляду для осознания наличия информации в картинке. После скачивания изображения из шага 4, в шаге 5 предлагается дешифровать, путем загрузки изображения из шага 4. В результате программа автоматически скачивает расшифрованный файл с информацией.



Рис. 1 – окно разработанного приложения

**Вывод**

В ходе лабораторной работы были получены знания принципах работы стеганографического метода на основе преобразования наименее значащих бит. При выполнении лабораторной работы было разработано браузерное приложение на js и html+css, которое позволяет скрывать файл в изображение и расшифровывать изображения в файл. Были изучены основные критерии и особенности стеганографии, выявлены и усвоены отличающие черты от киптографии.