

Цель работы

Реализовать текстовую стегосистему на основе изменения порядка следования маркеров конца строки CR/LF.

Описание алгоритма

Метод изменения порядка следования маркеров конца строки CR/LF использует индифферентность подавляющего числа средств отображения текстовой информации к порядку следования символов перевода строки (CR) и возврата каретки (LF), ограничивающих строку текста. Традиционный порядок следования CR/LF соответствует 0, а инвертированный LF/CR означает 1.

Ход работы

При формировании сообщения для передачи формируем контейнер. Если на позиции, соответствующей строке, в сообщении стоит 0, то оставляем порядок следования маркеров CR/LF. Если же в сообщении на позиции стоит 1, то следует инвертировать порядок следования маркеров CR/LF.

При приеме сообщения происходит декодирование, которое заключается в следующем: если строка начинается с CR, то есть с символа $\backslash r$, то декодируем её в 0, а если же начинается с LF, то есть с символа $\backslash n$, то декодируем её в 1.

В случае, если емкость контейнера больше передаваемого сообщения, то старшие разряды просто кодируются нулями, что никак не влияет на значение сообщения, так как это просто незначащие нули. (см. изображения 1, 2)

В случае, если емкость контейнера меньше передаваемого сообщения, мы передаем только часть, соответствующую размеру контейнера, а остальное игнорируем. (см. изображения 3, 4)

```

-----
                        Embedding mode
-----

Message to send:  [1 0 1]
Text:
The CR/LF line reordering method exploits
the indifference of a number of display media
to the ordering of line characters (CR)
and carriage return (LF) that delimit a line of text.
Traditional CR/LF order is 0, and inverted LF/CR is 1.
Message to send:  [1 0 1]
-----

                        Extract mode
-----

[0 0 1 0 1]

```

Рисунок 1. Передача произвольного сообщения длины 3.

```

1  The CR/LF line reordering method exploits
2
3  the indifference of a number of display media
4  to the ordering of line characters (CR)
5
6  and carriage return (LF) that delimit a line of text.
7  Traditional CR/LF order is 0, and inverted LF/CR is 1.
8  |

```

Рисунок 2. Файл с произвольным сообщением длины 3.

```

-----
                        Embedding mode
-----
Message to send:  [1 0 1 0 1 0 1 0 1 0 1 0 1 0 1]
Text:
The CR/LF line reordering method exploits
the indifference of a number of display media
to the ordering of line characters (CR)
and carriage return (LF) that delimit a line of text.
Traditional CR/LF order is 0, and inverted LF/CR is 1.
abc
def
ghi
Message to send:  [0 1 0 1 0 1 0 1]
-----
                        Extract mode
-----
[0 1 0 1 0 1 0 1]

```

Рисунок 3. Передача произвольного сообщения длины 15.

| | |
|----|--|
| 1 | The CR/LF line reordering method exploits |
| 2 | |
| 3 | the indifference of a number of display media |
| 4 | to the ordering of line characters (CR) |
| 5 | |
| 6 | and carriage return (LF) that delimit a line of text. |
| 7 | Traditional CR/LF order is 0, and inverted LF/CR is 1. |
| 8 | |
| 9 | abc |
| 10 | def |
| 11 | |
| 12 | ghi |
| 13 | |

Рисунок 4. Файл с произвольным сообщением длины 8.

Теоретические вычисления

- Файл формата txt:
 - 1) Размер пустого контейнера: 257 байт (8 строк)
 - 2) Максимальный размер сообщения для передачи: 8 бит
 - 3) Ёмкость контейнера:

$$\frac{8 \text{ бит внедренной информации}}{257 \text{ байт исходного сообщения}} = 0,031 \frac{\text{бит}}{\text{байт}}$$

Исходя из полученного значения, можно сделать вывод, что емкость контейнера достаточно низкая в конкретно этой ситуации.

- Файл формата html:

- 1) Размер пустого контейнера: 300 байт
- 2) Максимальный размер сообщения для передачи: 12 бит
- 3) Ёмкость контейнера:

$$\frac{12 \text{ бит внедренной информации}}{300 \text{ байт исходного сообщения}} = 0,040 \frac{\text{бит}}{\text{байт}}$$

Таким образом, в данной ситуации ёмкость при работе с форматом html выше, чем ёмкость в примере с txt файлом.

- Файл формата rtf:

- 1) Размер пустого контейнера: 224 байта
- 2) Максимальный размер сообщения для передачи: 7 бит
- 3) Ёмкость контейнера:

$$\frac{7 \text{ бит внедренной информации}}{224 \text{ байт исходного сообщения}} = 0,031 \frac{\text{бит}}{\text{байт}}$$

Исходя из полученного значения в данной конкретной ситуации ёмкость при работе с форматом rtf примерно совпадает с ёмкостью, полученной при работе с файлом формата txt.

Вывод

Таким образом, в ходе выполнения данной лабораторной работы была реализована стегосистема на основе изменения порядка следования маркеров конца строки CR/LF. Данный способ обладает малой пропускной способностью и не может использоваться в современных системах. Также стоит отметить, что для разных форматов файлов алгоритм работает по-разному.

Листинг программы

main.go:

```
package main

import (
    "fmt"
)
const LengthMessage = 100

func main() {
    fmt.Println("-----")
    --"
    fmt.Println("\t \t Embedding mode")
    fmt.Println("-----")
    --"

    message := createExampleMessage(LengthMessage)
    fileInput := openFile("inputFile.rtf")
    defer fileInput.Close()
    fileOutput := createFile("outputFile.rtf")
    embedMessageInFile(fileInput, message, fileOutput)
    fileOutput.Close()

    fmt.Println("-----")
    --"
    fmt.Println("\t \t Extract mode")
    fmt.Println("-----")
    --"
    fileOutput = openFile("outputFile.rtf")
    extractMessageFromFile(fileOutput)
    defer fileOutput.Close()
}
```

worker.go:

```
package main

import (
    "bufio"
    "fmt"
    "io"
    "os"
    "strings"
)

const CrLf = "\r\n"
const LfCr = "\n\r"

func createExampleMessage(length int) []byte {
    message := make([]byte, length)
```

```

    for i := 0; i < cap(message); i++ {
        if i%2 == 1 {
            message[i] = 0
        } else {
            message[i] = 1
        }
    }
    fmt.Println("Message to send: ", message)
    return message
}

func openFile(fileName string) *os.File {
    f, err := os.Open(fileName)
    if err != nil {
        fmt.Println(err.Error())
    }
    return f
}

func createFile(fileName string) *os.File {
    file, err := os.Create(fileName)
    if err != nil {
        fmt.Println("Error create file: ", err)
        panic(1)
    }
    return file
}

func embedMessageInFile(inputFile *os.File, message []byte,
    outputFile *os.File) {
    reader := bufio.NewReader(inputFile)
    indexOfMessage := len(message) - 1
    numberOfBits := 0
    fmt.Println("Text:")
    for {
        line, err := reader.ReadString('\n')
        if err != nil {
            if err == io.EOF {
                break
            } else {
                fmt.Println(err)
                return
            }
        }
        fmt.Print(line)
        result := embedMessageInString(line, message,
            indexOfMessage)
        _, err = outputFile.WriteString(result)
        if err != nil {
            fmt.Println(err.Error())
        }
    }
}

```

```

        if 0 <= indexOfMessage {
            indexOfMessage--
            numberOfBits++
        }
    }
    fmt.Println("Message to send: ", message[len(message)-
numberOfBits:])
}

func extractMessageFromFile(fileWithMessage *os.File) {
    reader := bufio.NewReader(fileWithMessage)
    var message []byte
    var text []string
    for {
        line, err := reader.ReadString('\n')

        if err != nil {
            if err == io.EOF {
                break
            } else {
                fmt.Println(err)
                return
            }
        }
        text = append(text, line)
    }
    for i := len(text) - 1; i >= 0; i-- {
        message = append(message, extractMessage(text[i]))
    }
    fmt.Println(message)
}

func embedMessageInString(containerString string, message
[]byte, bitCounter int) (resultString string) {
    resultString = containerString[:len(containerString)-2]
    if bitCounter >= 0 {
        if message[bitCounter] == 1 {
            resultString = resultString + LfCr
            return resultString
        } else {
            resultString = resultString + CrLf
            return resultString
        }
    } else {
        resultString = resultString + CrLf
        return resultString
    }
}

func extractMessage(containerString string) (partOfMessage
byte) {

```

```
    if strings.Contains(containerString, CrLf) {  
        return 0  
    } else {  
        return 1  
    }  
}
```