Отчёт по лабораторной работе №8

Элементы криптографии. Шифрование (кодирование) различных исходных текстов одним ключом

Косолапов Степан Эдуардович НПИбд-01-20

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Освоить на практике применение режима однократного гаммирования на примере кодирования различных исходных текстов одним ключом.

2 Выполнение работы

1. Код будем писать на языке Javascript. Напишем сперва вспомогательные фунцкии stringToHex и hexToString. Они будут переводить нам набор символов в шестнадцатеричные числа - соответствующие ASCII коду символа в кодировке UTF-16, разделённые пробелом:

```
const stringToHex = (str) => {
  return str.split('').map(c => c.charCodeAt(0).toString(16)).join(' ');
}

const hexToString = (hexStr) => {
  return hexStr.split(' ').map(c => String.fromCharCode(Number.parseInt(c, 16))).join('');
}
```

2. Напишем функцию, которая сгенерирует случайный набор символов(от 0 до 1048 в ASCII UTF-16) указанной длины, в шестнадцатеричном представлении:

```
const generateKey = (length) => {
  const result = [];

  for (let i = 0; i < length; i++) {
    const asciiCode = Math.floor(Math.random() * 1048);
    result.push(asciiCode.toString(16));
  }
```

```
    return result.join(' ');
}
```

3. Так же напишем основную функцию, которая и будет выполнять шифрование. Она принимает на вход два шестнадцатеричных набора(текст и ключ), и выполняет xor посимвольно, возвращая новый шестнадцатеричный набор:

```
const gammingCipher = (hexText, hexKey) => {
    const textSplit = hexText.split(' ');
    const keySplit = hexKey.split(' ');

    if (textSplit.length !== keySplit.length) {
        throw new Error('Key and message must have equal lengths.');
    }

    return textSplit.map((textCharHex, i) => {
        const keyCharHex = keySplit[i];

        const xorResult = Number.parseInt(textCharHex, 16) ^ Number.parseInt(keyCharHex, 16); // p_i xor k_i

        return xorResult.toString(16);
    }).join(' ')
}
```

4. Шифруем сообщения:

```
const message = 'Штирлиц – Вы Герой!!!!';
const message2 = 'Привет, Штирлиц, ура!!';

const hexMessage = stringToHex(message);
const hexMessage2 = stringToHex(message2);
```

```
const hexKey = '203 3e7 2ea ec 2dc 29 3b4 10b 7f 23 33b 185 1ac 121 26f 97 1d5 3ad 1a3 97 25a 3c1';

const enc1 = gammingCipher(hexMessage, hexKey);
const enc2 = gammingCipher(hexMessage2, hexKey);
```

5. Предположим мы знаем как выглядят оба сообщения, но в какой-то мере:

```
let known1 = 'Штирлиц**********!!!!';
let hexKnown1 = stringToHex(known1);

let known2 = '******, Штирлиц, *****';
let hexKnown2 = stringToHex(known2);
```

6. Проделаем known1 xor enc1 xor enc2 и заполняем known2 исходя из результата:

```
console.log('known1 xor enc1 xor enc2', hexToString(gammingCipher(enc2, gammingCipher(hexKnown1, enc1))))

known2 = 'Привет, Штирлиц, *ра!!';
hexKnown2 = stringToHex(known2);
console.log('known2:', known2);
```

7. Теперь, делаем known2 xor enc2 xor enc1 и заполняем known1 исходя из результата:

```
console.log('known2 xor enc2 xor enc1', hexToString(gammingCipher(enc1, gammingCipher(hexKnown2, enc2))))
known1 = 'Штирлиц – Вы Герой!!!!';
hexKnown1 = stringToHex(known1);
console.log('known1:', known1);
```

8. Теперь мы знаем полностью первый текст и можем легко найти второй текст:

```
console.log('known1 xor enc1 xor enc2', hexToString(gammingCipher(enc2, gammingCipher(hexKnown1, enc1))))
```

```
known1 = 'Штирлиц – Вы Герой!!!!';
known2 = 'Привет, Штирлиц, ура!!';

console.log('known1:', known1);
console.log('known2:', known2);
```

9. Результат работы программы:

```
node index.js

known1 Штирлиц**********!!!!
known2 ******, Штирлиц, *****

known1 xor enc1 xor enc2 Привет,*ш!бYцдPра!!
known2: Привет, Штирлиц, *ра!!

known2 xor enc2 xor enc1 Штирлиц – Вы ГероР!!!!
known1: Штирлиц – Вы Герой!!!!

known1 xor enc1 xor enc2 Привет, Штирлиц, ура!!
known1: Штирлиц – Вы Герой!!!!
known2: Привет, Штирлиц, ура!!
```

3 Контрольные вопросы

1. Если вы знаете один из исходных текстов, вы можете просто применить операцию XOR к известному тексту и соответствующему зашифрованному тексту. Результатом этой операции будет другой исходный текст. Это связано с свойствами операции XOR.

2. Если ключ повторно используется для шифрования другого текста, становится возможной атака с использованием метода, описанного в примере выше. Два разных текста, зашифрованных с использованием одного и того же ключа, могут быть совмещены с помощью операции XOR, чтобы получить XOR двух исходных текстов. Это может облегчить криптанализ и потенциальное расшифрование исходных текстов.

3. Режим однократного гаммирования реализуется применением операции XOR к каждому символу открытого текста с соответствующим символом секретного ключа. Это происходит последовательно для каждого символа двух текстов, используя один и тот же ключ.

4. Основной недостаток шифрования двух открытых текстов одним ключом - это уязвимость к атакам. Если злоумышленник получит два зашифрованных текста, которые были зашифрованы с использованием одного и того же ключа, то с использованием достаточно простой методики он сможет восстановить исходные тексты.

5. Одним из преимуществ шифрования двух текстов одним ключом является удобство - вам не нужно отслеживать, какой ключ использовался для каждого сообщения. Кроме того, если секретный ключ сохраняется в секрете, этот метод шифрования

может быть вполне простым и эффективным. Это также способ шифрования с низкими вычислительными затратами, что может быть полезно в некоторых ситуациях.

4 Выводы

В данной работе мы освоили на практике применение режима однократного гаммирования на примере кодирования различных исходных текстов одним ключом.