Politechnika Białostocka

Sprawozdanie z pracowni specjalistycznej

Bezpieczeństwo Sieci Komputerowych

Ćwiczenie numer: 3

Temat: Implementacja podstawowych modułów kryptograficznych

Wykonujący ćwiczenie:

Paweł Orzel

Łukasz Hossa

Kasper Seweryn

Studia dzienne

Kierunek: Informatyka

Semestr: VI

Grupa zajęciowa: Grupa PS 10

Prowadzący ćwiczenie: mgr inż. Katarzyna Borowska

Data wykonania ćwiczenia:07.03.2022

1.Treść zadania

Zaimplementuj i zaprezentuj działanie poniższych prostych metod ochrony danych poprzez szyfrowanie. Program powinien obsługiwać zarówno opcję szyfrowania, jak i deszyfrowania.

2.Teoria

Szyfry przestawieniowe polegają na tym, że zaszyfrowana wiadomość składa się ze wszystkich znaków występujących w szyfrze jednak w innej kolejności. Wiadomości odczytuje się z wykorzystaniem klucza za pomocą którego, w zależności od algorytmów jest odczytywana wiadomość. Z kolei w szyfrach afinicznych nie występują już litery z którego złożone jest słowo jawne, lecz każdej literze słowa jawnego odpowiada jedna z alfabetu zaszyfrowanego.

3.Opis realizacji wymagań

Zostały spełnione wszystkie wymagania. Zaimplementowane zostało 6 algorytmów szyfrujących zgodnie z wymaganiami prowadzącego. Za ich pomocą można zarówno zakodować jak i odkodować wiadomość.

4.Zastosowane technologie

Algorytmy zostały zaimplementowane w języku TypeScript, a za pomocą frameworku Vue został stworzony interfejs graficzny do łatwego posługiwania się algorytmami.

5.Działanie programu

6.Rozwiązanie problemów

1.Rail Fence

Testy

```
test('encrypt with n = 3', async () => {
  const n = 3
  expect(encrypt(message, n)).toEqual('CTARPORPYYGH')
})

test('encrypt with n = 4', async () => {
  const n = 4
  expect(encrypt(message, n)).toEqual('CGRORYYTAHPP')
})

test('encrypt with n = 5', async () => {
  const n = 5
  expect(encrypt(message, n)).toEqual('CARRPYGHPOYT')

})
```

```
test('decrypt with n = 3', async () => {
  const n = 3
  expect(decrypt('CTARPORPYYGH', n)).toEqual(message)
})

test('decrypt with n = 4', async () => {
  const n = 4
  expect(decrypt('CGRORYYTAHPP', n)).toEqual(message)
})

test('decrypt with n = 5', async () => {
  const n = 5
  expect(decrypt('CARRPYGHPOYT', n)).toEqual(message)

})
```

```
√ test/railFence.test.ts (8)
```

Omówienie kodu

Szyfrowanie

```
const encrypt = (message: string, rows: number) => {
  const res: string[][] = [...Array(rows).keys()].map(() => [])
  for (let i = 0; i < message.length; ++i) {
    const k = i % (rows * 2 - 2)
    res[k < rows ? k : rows - (k % rows + 1) - 1][i] = message[i]
  }

  return res.flat().join('')
```

Tworzona jest pusta macierz, którą następnie wypełniamy literami w kształt „płotka” i odczytujemy zakodowane słowo wierszami

Deszyfrowanie

```
let index = 0
for(let i = 0; i < rows; i++){
  for(let j = 0; j < message.length; j++){
    if(res[i][j] === '*' && index < message.length){
      res[i][j] = message[index];
      index = index + 1
```

```
for(let i = 0; i < message.length; i++){
  if(row === 0){
    directory = true
  }
  if(row === rows - 1){
    directory = false;
  }

  if(res[row][col] != '*'){
    result.push(res[row][col])
    col++
  }

  if(directory){
    row++
  }
  else{
    row--
  }
```

Gwiazdkami wyznaczamy schemat „płotkowy” i wypełniamy gwiazdki literami zakodowanego słowa wiersz po wierszu. Następnie litery są odczytywane w kierunku płotkowym za pomocą wyznaczonego kierunku i dodawane są do odkodowanego słowa

2.Przestawienia macierzowe 2a

Testy

```ts
test('decrypt with 2 column key', async () => {
  const key = '2-1'

  expect(decrypt('RCPYOTRGPAYHSOA', key)).toEqual(message)
})

test('decrypt with 4 column key', async () => {
  const key = '3-1-4-2'

  expect(decrypt('YCPRGTROHAYPAOS', key)).toEqual(message)
})

test('decrypt with 5 column key', async () => {
  const key = '3-4-1-5-2'

  expect(decrypt('YPCTRRAOPGOSHAY', key)).toEqual(message)
})
```

```ts
const message = 'CRYPTOGRAPHYOSA'

test('encrypt with 2 column key', async () => {
  const key = '2-1'

  expect(encrypt(message, key)).toEqual('RCPYOTRGPAYHSOA')
})

test('encrypt with 4 column key', async () => {
  const key = '3-1-4-2'

  expect(encrypt(message, key)).toEqual('YCPRGTROHAYPAOS')
})

test('encrypt with 5 column key', async () => {
  const key = '3-4-1-5-2'

  expect(encrypt(message, key)).toEqual('YPCTRRAOPGOSHAY')
```

```
√ test/matrixACipher.test.ts (6)
```

Omówienie

Szyfrowanie

```ts
const encrypt = (message: string, key: string) => {
  if (!isKeyValid(key)) throw new Error('Invalid key')

  const indexes = key.split('-').map((i: any) => i - 1)
  const matrix = messageToMatrix(message, indexes.length)
  return transpose(indexes.map(i => matrix[i])).flat().join('')
}
```

Przestawiamy kolumny w odpowiedniej kolejności (wyznaczonej przez indeksy z klucza) po czym robimy transpozycje macierzy i sczytujemy rzędami.

Deszyfrowanie

```ts
const result = []
for (const line of transposed) {
  // NOTE: We're removing the biggest indexes from the line to fix the uneven last line
  const skip = indexes.map(i => i).sort((a, b) => b - a)
  skip.length = indexes.length - line.length
  const filteredIndexes = indexes.filter(i => !skip.includes(i))

  const decryptedRow = filteredIndexes
    .map((strength, i) => ({ strength, char: line[i] }))
    .sort((a, b) => a.strength - b.strength)
    .map(({ char }) => char)

  result.push(decryptedRow)
```

Tworzymy macierz ze słowa, następnie dokonujemy transpozycji dzięki której możemy pracować na rzędach zamiast na kolumnach, obliczamy ile znaków brakuje do pełnego rzędu i wyrzucamy tyle największych indeksów z klucza, następnie przestawiamy wiersz wedle

indeksów w nowopowstałym kluczu. Na koniec sczytujemy wartość rzędami.

3.Przestawienia macierzowe 2b

Testy

```
const message = 'HERE IS A SECRET MESSAGE ENCIPHERED BY TRANSPOSITION'

test('encrypt to key CONVENIECE', async () => {
  const key = 'CONVENIENCE'
  expect(encrypt(message, key)).toEqual('HECRNCEYIISEPSGDIRNTOAAESRMPNSSROEEBTETIAEEHS')
})

test('decrypt to key CONVENIENCE', async () => {
  const key = 'CONVENIENCE'
  expect(decrypt('HECRNCEYIISEPSGDIRNTOAAESRMPNSSROEEBTETIAEEHS', key))
    .toEqual(message.replace(/ +/g, ''))
})

test('encrypt to key POLITECHNIKA', async () => {
  const key = 'BIALYSTOK'
  expect(encrypt(message, key)).toEqual('REEDOHCGRSEREEPEAENNETNBSSSHAOSEITTASPRIIMCYI')
})


test('decrypt to key POLITECHNIKA', async () => {
  const key = 'BIALYSTOK'
  expect(decrypt('REEDOHCGRSEREEPEAENNETNBSSSHAOSEITTASPRIIMCYI', key))
    .toEqual(message.replace(/ +/g, ''))
})
test('encrypt with message shorter than key', async () => {
  expect(encrypt('CONVENIENC', key)).toEqual('CCEEINNNOV')
})

test('decrypt with message shorter than key', async () => {
  // NOTE: Due to the message being shorter than the key, the output cannot be deciphered correctly
  expect(decrypt('CCEEINNNOV', key)).toEqual('CVNENNEOCI')
})
```

```
√ test/columnarTranspositionCipher.test.ts (4)
```

Omówienie

Szyfrowanie

```
const encrypt = (message: string, key: string) => {
  const indexes = key.toUpperCase().split('')
    .map((char, index) => ({ char, index, code: char.charCodeAt(0) }))
    .sort((a, b) => a.code - b.code)
    .map(entry => entry.index)

  const matrix = messageToMatrix(message.replace(/ +/g, ''), indexes.length)
  return indexes.map(i => matrix[i]).flat().join('')
}
```

Każdy znak klucza jest zamieniony na obiekt, który
przechowuje ten znak, kod char code tego znaku oraz oryginalny indeks w
kluczu, następnie jest sortowany według char code i zmapowany na oryginalny indeks.
Dzięki temu później możemy po prostu zmapowac te indeksy na
odpowiadające im kolumny z macierzy, a następnie odczytać wartości
Wierszami. Nie dodaliśmy spacji w wyniku szyfrowania, aby utrudnić odgadnięcie klucza

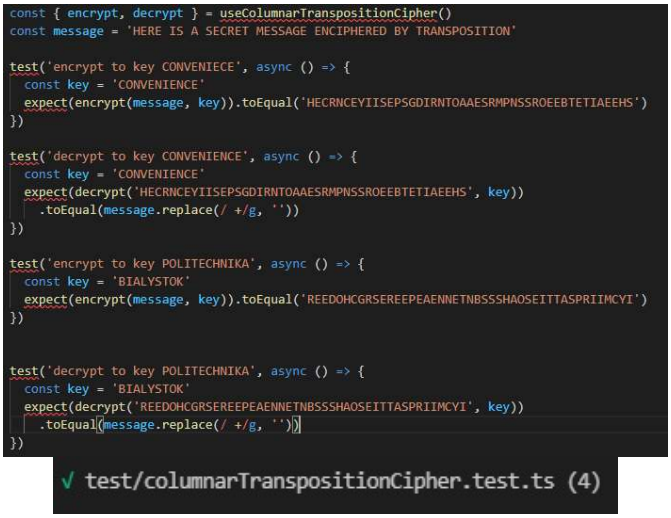
Deszyfrowanie

```
const decrypt = (message: string, key: string) => {
 const indexes = key.toUpperCase().split('')
   .map((char, index) => ({ char, index, code: char.charCodeAt(0) }))
   .sort((a, b) => a.code - b.code)
   .map((entry, index) => ({ ...entry, strength: index }))
   .sort((a, b) => a.index - b.index)
   .map(entry => entry.strength)

 const lengths = messageToMatrix(message.replace(/ +/g, ''), indexes.length)
   .map((arr: string[][]) => arr.length)

 const letters = message.replace(/ +/g, '').split('')
 const columns = lengths.map((length: number) => letters.splice(0, length)).map((col: string[]) => col.join(''))
 return transpose(indexes.map(i => columns[i] ? columns[i].split('') : [])).flat().join('')
}
```

Każdy znak klucza jest zamieniony na obiekt, który przechowuje ten znak, kod char code tego znaku oraz oryginalny indeks w kluczu, następnie jest sortowany wg char code. Dodajemy do tego obiektu aktualny indeks po sortowaniu jako sile danego znaku i sortujemy po oryginalnych indeksach po czym mapujemy obiekt na jego sile. Następnie, aby znaleźć oryginalne długości słów wpisujemy wiadomość w macierz po czym mapujemy każdy element na długość otrzymanego słowa. Po czym rozbijamy oryginalna wiadomość na słowa z nowo otrzymanych długości i sczytujemy wierszami macierz utworzona z ze słów ustawionych w kolejności w jakiej mówiły nasze indeksy na której dokonano transpozycji.

4.Przestawienia macierzowe 2c

Testy



```
const { encrypt, decrypt } = useColumnarTranspositionCipher()
const message = 'HERE IS A SECRET MESSAGE ENCIPHERED BY TRANSPOSITION'

test('encrypt to key CONVENIECE', async () => {
  const key = 'CONVENIENCE'
  expect(encrypt(message, key)).toEqual('HECRNCEYIISEPSGDIRNTOAAESRMPNSSROEEBTETIAEEHS')
})

test('decrypt to key CONVENIENCE', async () => {
  const key = 'CONVENIENCE'
  expect(decrypt('HECRNCEYIISEPSGDIRNTOAAESRMPNSSROEEBTETIAEEHS', key))
    .toEqual(message.replace(/ +/g, ''))
})

test('encrypt to key POLITECHNIKA', async () => {
  const key = 'BIALYSTOK'
  expect(encrypt(message, key)).toEqual('REEDOHCGRSEREEPEAENNETNBSSSHAOSEITTASPRIIMCYI')
})


test('decrypt to key POLITECHNIKA', async () => {
  const key = 'BIALYSTOK'
  expect(decrypt('REEDOHCGRSEREEPEAENNETNBSSSHAOSEITTASPRIIMCYI', key))
    .toEqual(message.replace(/ +/g, ''))
})
```

```
√ test/columnarTranspositionCipher.test.ts (4)
```

Omówienie

Szyfrowanie

```ts
const encrypt = (message: string, key: string) => {
  const indexes = createIndexes(key)
  const table = createTable(message, indexes)

  const matrix = transpose(table)
  return indexes.map(i => matrix[i.index])
    .map((i: string[]) => i ? i.join('') : '').join('')
}
```

```ts
const createIndexes = (key: string) => key.toUpperCase().split('')
  .map((char, index) => ({ char, index, code: char.charCodeAt(0) }))
  .sort((a, b) => a.code - b.code)
  .map((entry, index) => ({ ...entry, strength: index }))

const createTable = (message: string, indexes: { index: number }[]) => {
  const table: string[] = ['']
  for (const char of message.replace(/ +/g, '')) {
    const i = table.length - 1
    const { index } = indexes[i]

    table[i] += char
    if (index + 1 === table[i].length) {
      table.push('')
    }
  }

  return table.map(str => str.split(''))
}
```


```ts
const createTable = (message: string, indexes: { index: number }[]) => {
  const table: string[] = ['']
  for (const char of message.replace(/ +/g, '')) {
    const i = table.length - 1
    const { index } = indexes[i]

    table[i] += char
    if (index + 1 === table[i].length) {
      table.push('')
    }
  }

  return table.map(str => str.split(''))
}
```


Wpierw tworzymy indeksy podobnie jak podczas szyfrowania w poprzednim algorytmie, natomiast zamiast mapować na oryginalny indeks dodajemy do obiektów ich sile. Następnie Tworzymy tabele wpisując odpowiednio kolejne znaki tak, aby w rządku było maksymalnie tyle znaków jaka jest siła obiektu dla którego indeks aktualnego rządku odpowiada indeksowi tego obiektu. Potem dokonujemy transpozycji na tabeli i mapujemy oryginalne indeksy obiektów na kolumny nowej macierzy po czym sczytujemy wiersz po wierszu.

Deszyfrowanie

```
const decrypt = (message: string, key: string) => {
  const indexes = createIndexes(key)

  const table = createTable(message, indexes)
  if (!table.slice(-1)[0].length) table.pop()

  const letters = message.split('')
  const wordLengths = transpose(table)
    .map((arr: string[]) => arr.reduce((acc, e) => acc + +(e !== null), 0))

  const words = indexes.map(i => wordLengths[i.index]) .map(length => letters.splice(0, length).join(''))
  for (const [x, word] of Object.entries(words)) {
    let y = 0
    for (const char of word) {
      const { index } = indexes[+x]
      while (!table[y][index]) y += 1
      table[y][index] = char
      y += 1
    }
  }

  return table.flat().join('')
```

Tworzymy indeksy i tabele tak samo jak w przypadku szyfrowania. Tworzenie tabeli może dodać nam pojedynczy pusty rządek w przypadku gdy ostatni wiersz jest dopełniony. Jeżeli takowy wystąpił to go usuwamy. Dzieki tablicy jestesmy w stanie znalezc oryginalne długości słów i podzielić oryginalną wiadomość na słowa tej długości. Następnie iterujemy po każdym słowie i każdym znaku podmieniając odpowiednie litery w tabeli na litery ze znaku. Na końcu sczytujemy wiersz po wierszu.

5.Szyfr Cezara

Testy

```
const message = 'CRYPTOGRAPHY'

test('encrypt with k = 2', async () => {
  const key = 2

  expect(encrypt(message, key)).toEqual('ETARVQITCRJA')
})

test('encrypt with k = 3', async () => {
  const key = 3

  expect(encrypt(message, key)).toEqual('FUBSWRJUDSKB')
})

test('encrypt with k = 4', async () => {
  const key = 4

  expect(encrypt(message, key)).toEqual('GVCTXSKVETLC')
})

test('encrypt with k = 5', async () => {
  const key = 5

  expect(encrypt(message, key)).toEqual('HWDUYTLWFUMD')
})
```

```
test('decrypt with k = 2', async () => {
  const key = 2

  expect(decrypt('ETARVQITCRJA', key)).toEqual(message)
})

test('decrypt with k = 3', async () => {
  const key = 3

  expect(decrypt('FUBSWRJUDSKB', key)).toEqual(message)
})

test('decrypt with k = 4', async () => {
  const key = 4

  expect(decrypt('GVCTXSKVETLC', key)).toEqual(message)
})

test('decrypt with k = 5', async () => {
  const key = 5

  expect(decrypt('HWDUYTLWFUMD', key)).toEqual(message)
})
```

```
√ test/caesarCipher.test.ts (8)
```

Omówienie

Szyfrowanie

```
function mod(n: number, m: number) {
return ((n % m) + m) % m;
}


const encrypt = (message: string, key: number) => {
    const alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    let result = ''
    message = message.toUpperCase();
    for(let i = 0 ; i < message.length; i++){
        let index = alphabet.indexOf(message[i])

        result = result + alphabet[mod((+index + +key),26)]

    }
    return result;

  }
```

Szyfrowanie odbywa się za pomocą podanego wzoru w treści zadania. Dzięki operacji modulo i dodawaniu klucza szyfrujemy wiadomość za pomocą przesuwania literek w alfabecie o długość podanego klucza.

Deszyfrowanie

Deszyfracja odbywa się za pomocą funkcji szyfrującej, jednak w tym przypadku klucz jest mnożony przez -1. Własna funkcja modulo dla liczb ujemnych pozwala obliczać przesunięcie klucza w lewo i względem wcześniejszej szyfracji przesuniemy się w to samo miejsce czym odszyfrowujemy wiadomość

6. Szyfrowanie Vigenere'a

Testy

```
const message = 'CRYPTOGRAPHY'

test('encrypt1', async () => {
  const key = 'BREAKBREAKBR'
  expect(encrypt(message, key)).toEqual('DICPDPXVAZIP')
})

test('decrypt1', async () => {
  const key = 'BREAKBREAKBR'
  expect(decrypt('DICPDPXVAZIP', key)).toEqual(message)
})

test('encrypt2', async () => {
  const key = 'POLITECHNIKA'
  expect(encrypt(message, key)).toEqual('RFJXMSIYNXRY')
})

test('decrypt2', async () => {
  const key = 'POLITECHNIKA'
  expect(decrypt('RFJXMSIYNXRY', key)).toEqual(message)
})
```

```
test('encrypt3', async () => {
  const key = 'POLITECHNIK'
  expect(encrypt(message, key)).toEqual('RFJXMSIYNXRN')
})

test('decrypt3', async () => {
  const key = 'POLITECHNIK'
  expect(decrypt('RFJXMSIYNXRN', key)).toEqual(message)
})

test('encrypt4', async () => {
  const key = 'POLITECHNIKAA'
  expect(encrypt(message, key)).toEqual('RFJXMSIYNXRY')
})

test('decrypt4', async () => {
  const key = 'POLITECHNIKAA'
  expect(decrypt('RFJXMSIYNXRY', key)).toEqual(message)
})
```

```
√ test/vigenere.test.ts (8)
```

Omówienie

Szyfrowanie

```
function encrypt (message: string, key: string) {
  let result = ''
  let keyOffset = 0

  for (const char of message.toUpperCase()) {
    if (!isLetter(char)) {
      result += char
      continue
    }

    const keyCharCode = key.toUpperCase().charCodeAt(keyOffset++)
    keyOffset %= key.length

    result += String.fromCharCode((char.charCodeAt(0) + keyCharCode - 2 * CHARCODE_A) % 26 + CHARCODE_A)
  }

  return result
}
```

Szyfrowanie przebiega podobnie jak w szyfrze Cezara. Liczba modulo długości alfabetu z sumy litery słowa z literą klucza da nam zaszyfrowaną literę.

Deszyfrowanie

Deszyfrowanie także jest podobne jak w szyfrze Cezara. Szukamy odszyfrowanych literek za pomocą liczby modulo różnicy literki zakodowanego słowa i literki klucza.

Algorytm działa także w przypadku, gdy długość słowa oraz klucza nie zgadzają się

7.Wnioski

Udało się zrealizować wszystkie zadania z algorytmów szyfrujących. Zostało napisane ponad 40 testów, które pomogły ocenić prawidłowość działania zaimplementowanych algorytmów. Stworzony interfejs graficzny jest bardzo intuicyjny i prosty. Według autorów niektóre algorytmy wykazują się większą ochroną zakodowanej wiadomości i są trudniejsze do złamania, aktualnie żaden z tych szyfrów nie jest bezpieczny. Przepływ pracy został udokumentowany za pomocą commitów w repozytorium github(https://github.com/pb-students/BSK-01-ciphers). Praca z algorytmami szyfrującymi pozwoliła na zrozumienie jak pomaga to w bezpieczeństwie haseł lub ważnych wiadomości