ZADANIA

BEZPIECZEŃSTWO SYSTEMÓW INFORMATYCZNYCH

PATRYK LACH 15415

Spis treści

1. Wstęp

W ramach zajęć laboratoryjnych stworzyłem aplikację do szyfrowania i deszyfrowania tekstów wykorzystującą różne algorytmy kryptograficzne. W ramach tej aplikacji zaimplementowano pięć popularnych szyfrów:

- Szyfr Cezara (Caesar Cipher) - Prosty szyfr, który polega na przesunięciu liter w alfabecie o określoną liczbę pozycji.
- Szyfr Polibiusza (Polybius Square) - Szyfr oparty na kwadracie 5x5, w którym każda litera jest reprezentowana przez dwie cyfry.
- Szyfr Vigenère'a (Vigenère Cipher) - Szyfr, który wykorzystuje powtarzający się klucz do przesunięcia liter w tekście.
- Szyfr Playfair (Playfair Cipher) - Szyfr, który operuje na parach liter, a tekst jest szyfrowany przy użyciu tablicy 5x5.
- Szyfr RSA (RSA Encryption) - Algorytm kryptograficzny oparty na kluczach publicznych i prywatnych, wykorzystywany do bezpiecznej wymiany danych.

Kod źródłowy aplikacji: https://github.com/taczhed/security-lab

2. Szyfr Cezara

Szyfr Cezara to prosty algorytm szyfrowania, który przesuwa każdą literę w tekście o stałą liczbę pozycji w alfabecie.

Funkcja: CaesarCipher(text, key)

- text: Ciąg znaków, który ma zostać zaszyfrowany lub odszyfrowany.
- key: Wartość przesunięcia, gdzie dodatnie wartości są używane do szyfrowania, a ujemne do odszyfrowania.

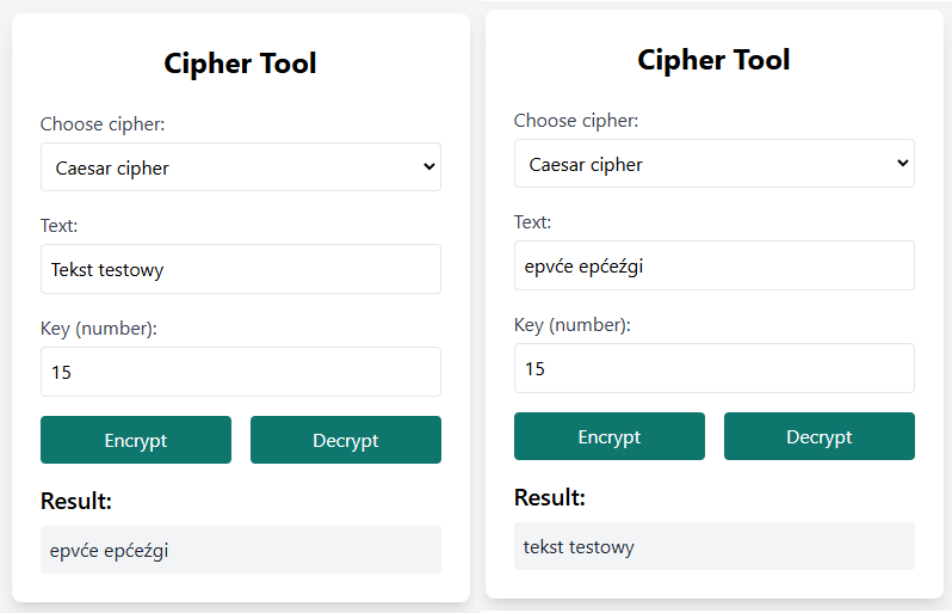
Zwracany wynik: Zaszyfrowany lub odszyfrowany tekst.

Kluczowa logika:

- Funkcja znajduje pozycję każdego znaku w dictionary.
- Następnie przesuwa znak o wartość podaną w key, obsługując zawijanie przy użyciu arytmetyki modulo.
- Znaki nienależące do alfabetu (np. spacje, interpunkcja) są zwracane bez zmian.

```
export const CaesarCipher = (text, key) => {  Show usages   taczhed
    const dictionary : string  = 'aąbcćdeęfghijklłmnńoópqrsśtuvwxyzźż';
    return text.split('').map((char) => {
        const index : number  = dictionary.indexOf(char.toLowerCase());
        if (index === -1) return char.toLowerCase();
        const shiftedIndex : number  = (index + (key % dictionary.length) + dictionary.length) % dictionary.length;
        return dictionary[shiftedIndex];
    }).join('');
};
```

Rysunek 1 - Kod



Rysunek 2 – Działanie

3. Szyfr Polibiusza

Szyfr Polibiusza to klasyczny algorytm kryptograficzny, który używa kwadratu 5x5 do reprezentacji liter. Tekst jest szyfrowany i deszyfrowany za pomocą współrzędnych w tym kwadracie.

Funkcja: PolybiusSquare(text, a, b)

- text: Ciąg znaków, który ma zostać zaszyfrowany lub odszyfrowany.
- a: Parametr klucza wykorzystywany do szyfrowania i deszyfrowania.
- b: Kolejny parametr klucza wykorzystywany do szyfrowania i deszyfrowania.

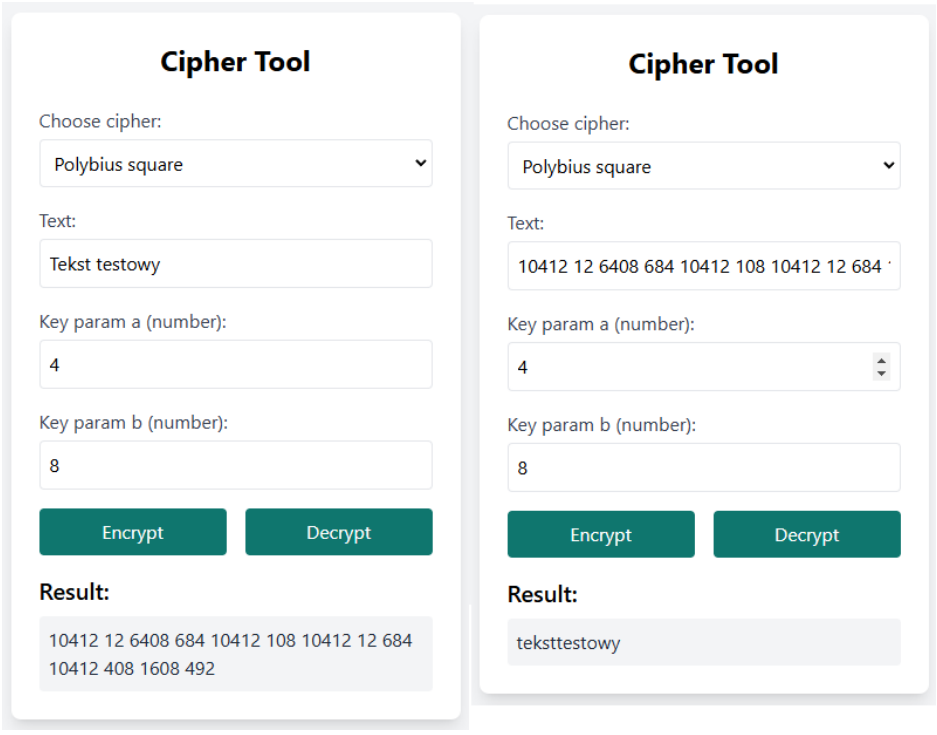
Zwracany wynik: Zaszyfrowany lub odszyfrowany tekst.

Kluczowa logika:

- Szyfrowanie:
    - Szyfrowanie polega na przypisaniu każdej literze tekstu jej współrzędnych w tablicy.
    - Następnie te współrzędne są szyfrowane za pomocą wzoru: a * x^2 + b.
    - Współrzędne są zamieniane na liczby, które są następnie szyfrowane.
- Deszyfrowanie:
    - Jeśli tekst składa się tylko z liczb, to traktujemy go jako zaszyfrowany tekst.
    - Deszyfrowanie polega na odczytaniu współrzędnych z tablicy dictionary, gdzie każda liczba jest przypisana do konkretnej litery.
    - Wykorzystuje się wzór do obliczenia współrzędnych liter z zaszyfrowanych liczb.

```
export const PolybiusSquare = (text, a, b) => {  Show usages  taczhed
    const dictionary : (...)[]  = [
        ['l', 'o', 'w', 'ź', 'k', 'u', 'ę'],
        ['e', 'y', 'ą', 'b', 'v', 't', 'h'],
        ['q', 'p', 'ń', 'd', 'f', 'm', 'g'],
        ['ć', 's', 'c', 'ł', 'z', 'x', 'a'],
        ['ó', 'j', 'ż', 'i', 'n', 'r', 'ś'],
        [' ']
    ];

    if (text.split(' ').every((e) => !isNaN(Number(e)))) {
        const decrypt = (y) => Math.sqrt( x: (y - b) / a);  Show usages  taczhed
        const firstStep : string[]  = text.split(' ').map((number) => {
            const de : number  = decrypt(number)
            const firstIndex : number  = Math.round( x: de / 10);
            const secondIndex : number  = de % 10;
            return dictionary[secondIndex][firstIndex]
        })
        return firstStep.join('')
    }

    const encrypt = (x) => a * Math.pow(x,  y: 2) + b;  Show usages  taczhed
    const firstStep : number[]  = text.split('').map((char) => {
        let res : number  = -1
        dictionary.forEach((yArray : string[] , y : number ) => {
            yArray.forEach((el : string , x : number ) => {
                if (el.toLowerCase() === char.toLowerCase()) res = Number( value: `${x}${y}`)
            })
        })
        return res
    })
    const secondStep : any[]  = firstStep.filter((n : number ) => n !== -1).map((t : number ) => encrypt(t))
    return secondStep.join(' ')
};
```

Rysunek 3 - Kod



Rysunek 4 – Działanie

4. Szyfr Vigenère'a

Szyfr Vigenère'a to klasyczny szyfr z wykorzystaniem klucza, który jest powtarzany, aby przesunąć litery w tekście o odpowiednie liczby pozycji, zależnie od liter klucza. Jest bardziej zaawansowanym rozwiązaniem w porównaniu do szyfru Cezara, ponieważ do szyfrowania każdej litery wykorzystuje inną wartość przesunięcia.

Funkcja: VigenereCipher(text, key, mode)

- text: Ciąg znaków, który ma zostać zaszyfrowany lub odszyfrowany.
- key: Klucz szyfrowania, który jest powtarzany do długości tekstu.
- mode: Tryb operacji.

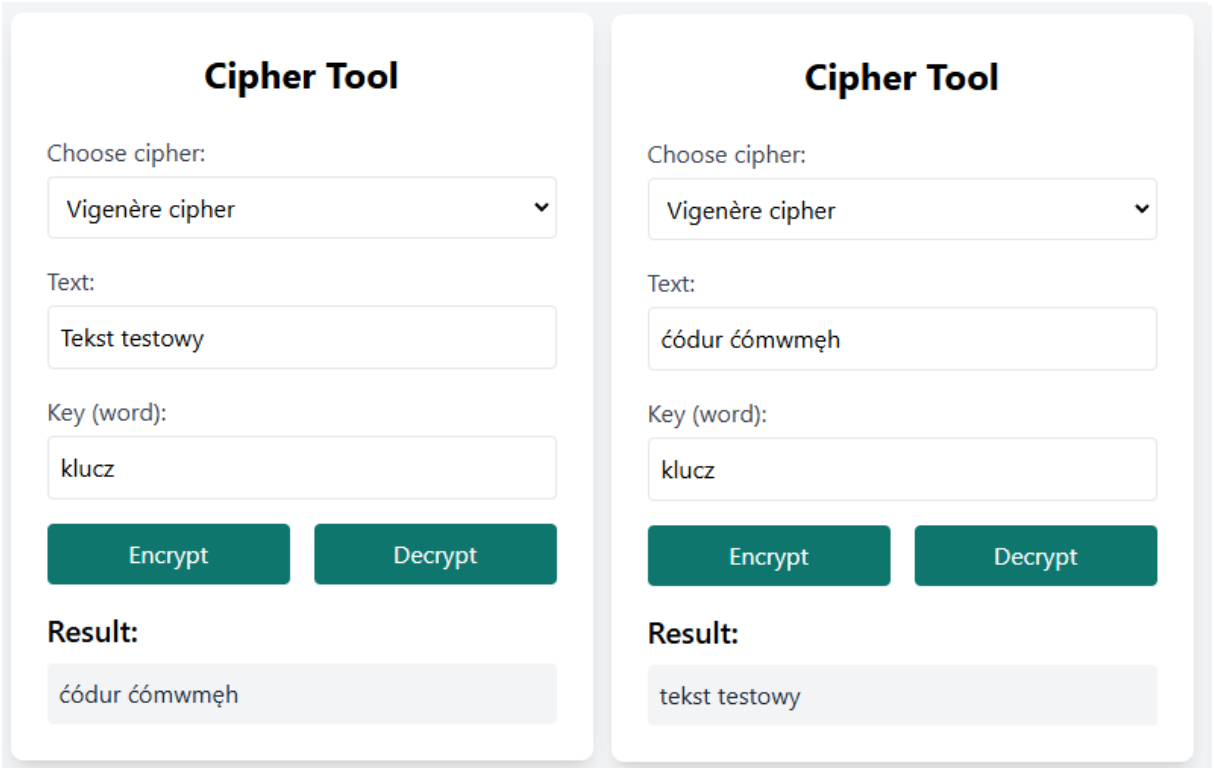
Zwracany wynik: Zaszyfrowany lub odszyfrowany tekst.

Kluczowa logika:

- Szyfrowanie:
    - Dla każdej litery w tekście, obliczane jest jej przesunięcie w zależności od odpowiadającej litery klucza.
    - Przesunięcie polega na dodaniu indeksu litery klucza do indeksu litery w tekście (modulo długości alfabetu).
- Odszyfrowanie:
    - W trybie odszyfrowania obliczane jest przesunięcie odwrotne, tj. odjęcie indeksu litery klucza od indeksu litery w tekście (modulo długości alfabetu).
- Zawijanie klucza:
    - Klucz jest powtarzany w razie potrzeby, aby pasował do długości tekstu, zapewniając, że każda litera w tekście będzie miała przypisany odpowiadający znak z klucza.
- Znaki spoza alfabetu:
    - Wszystkie znaki, które nie znajdują się w alfabecie (np. spacje, interpunkcja), są pozostawiane bez zmian.

```
export const VigenereCipher = (text, key, mode) => {  Show usages   taczhed
    const alphabet : string  = 'aąbcćdeęfghijklłmnńoópqrsśtuvwxyzźż';
    let keyIndex : number  = 0;
    key = key.toLowerCase();

    return text.toLowerCase().split( separator: '').map((char : string ) => {
        const index : number  = alphabet.indexOf(char);

        if (index !== -1) {
            const keyChar = key[keyIndex % key.length];
            const keyCharIndex : number  = alphabet.indexOf(keyChar);
            let newIndex;

            if (mode === 'encrypt') {
                newIndex = (index + keyCharIndex) % alphabet.length;
            } else if (mode === 'decrypt') {
                newIndex = (index - keyCharIndex + alphabet.length) % alphabet.length;
            }

            keyIndex++;
            return alphabet[newIndex];
        } else {
            return char;
        }
    }).join('')
}
```

Rysunek 5 - Kod



Rysunek 6 – Działanie

5. Szyfr Playfair'a

Szyfr Playfair'a to jeden z klasycznych szyfrów, który operuje na parach liter. Wykorzystuje on kwadrat 6x6 (lub 5x5 w wersji klasycznej) do szyfrowania tekstu, tworząc różne zasady dla par liter znajdujących się w tych samych wierszach, kolumnach lub w innych miejscach.

Funkcja: PlayfairCipher(text, key, mode)

- text: Ciąg znaków, który ma zostać zaszyfrowany lub odszyfrowany.
- key: Klucz, który jest używany do stworzenia kwadratu 6x6.
- mode: Tryb operacji.

Zwracany wynik: Zaszyfrowany lub odszyfrowany tekst.

Kluczowa logika:

- Przygotowanie tekstu:
    - Tekst jest konwertowany na małe litery, a następnie usuwane są wszystkie znaki, które nie są literami alfabetu.
    - Jeśli para liter w tekście jest identyczna (np. "ss"), dodawany jest znak 'x', aby rozdzielić je.
    - Jeśli tekst ma nieparzystą liczbę znaków, dodawany jest dodatkowy 'x', aby doprowadzić długość do parzystej.
- Tworzenie kwadratu:
    - Kwadrat 6x6 jest tworzony na podstawie klucza i alfabetu, który jest używany do szyfrowania.
    - Wszystkie powtarzające się litery w kluczu i alfabecie są usuwane, a pozostałe litery są umieszczane w kwadracie.
- Zasady szyfrowania:
    - Ta sama kolumna: Jeśli obie litery pary znajdują się w tej samej kolumnie, zamienia się je na litery znajdujące się w tej samej kolumnie, ale w wierszach sąsiednich (zawijając).
    - Ten sam wiersz: Jeśli obie litery pary znajdują się w tym samym wierszu, zamienia się je na litery znajdujące się w sąsiednich kolumnach.
    - Inne położenie: Jeśli litery znajdują się w różnych wierszach i kolumnach, zamieniają się miejscami w odpowiednich rogach prostokąta.
- Szyfrowanie i deszyfrowanie:
    - Szyfrowanie: W trybie szyfrowania, litery są zamieniane w sposób opisany powyżej, aby uzyskać zaszyfrowany tekst.
    - Deszyfrowanie: W trybie deszyfrowania, zasady są odwrotne (litery są przesuwane w przeciwnym kierunku w wierszach i kolumnach).

```
export const PlayfairCipher = (text, key, mode) => {  Show usages  taczhed
    text = prepareText(text);

    const modeValue : number  = mode === 'decrypt' ? -1 : 1;
    const alphabet : string  = 'aąbcćdeęfghijklłmnńoópqrsśtuvwxyzźż-';
    const grid : any[]  = createGrid(key, alphabet);
    let result : string  = '';

    for (let i : number  = 0; i < text.length; i += 2) {
        const pair : string  = text.slice(i, i + 2);
        const [row1 : number , col1 : number ] = findPosition(pair[0], grid);
        const [row2 : number , col2 : number ] = findPosition(pair[1], grid);

        if (row1 === row2) {
            // Same row: shift columns
            result += grid[row1][(col1 + modeValue + grid[row1].length) % grid[row1].length];
            result += grid[row2][(col2 + modeValue + grid[row2].length) % grid[row2].length];
        } else if (col1 === col2) {
            // Same column: shift rows
            result += grid[(row1 + modeValue + grid.length) % grid.length][col1];
            result += grid[(row2 + modeValue + grid.length) % grid.length][col2];
        } else {
            // Rectangle rule: swap columns
            result += grid[row1][col2];
            result += grid[row2][col1];
        }
    }
    return result;
};
```

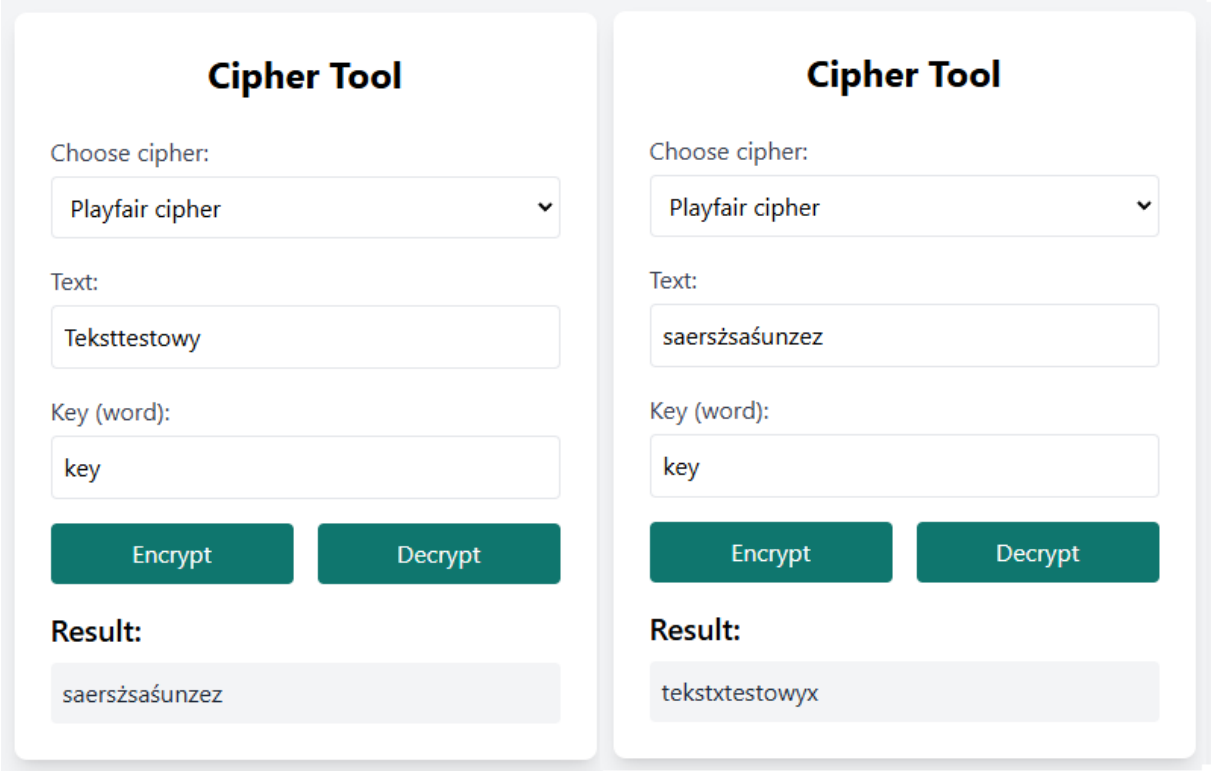
Rysunek 7 - Kod

```
const createGrid = (key, alphabet) => {  Show usages  taczhed
    key = key.toLowerCase().replace( searchValue: /[^aąbcćdeęfghijklłmnńoópqrsśtuvwxyzźż]/g,  replaceValue: '');
    let grid : any[]  = [];
    let combined : string  = key + alphabet;
    for (let char : string  of combined) {
        if (!grid.includes(char)) {
            grid.push(char);
        }
    }

    // 6 x 6
    const gridMatrix : any[]  = [];
    for (let i : number  = 0; i < grid.length; i += 6) {
        const row : any[]  = [];
        for (let j : number  = i; j < i + 6 && j < grid.length; j++) {
            row.push(grid[j]);
        }
        gridMatrix.push(row);
    }

    console.log(gridMatrix);

    return gridMatrix;
};

const findPosition = (char, grid) => {  Show usages  taczhed
    for (let row : number  = 0; row < grid.length; row++) {
        for (let col : number  = 0; col < grid[row].length; col++) {
            if (grid[row][col] === char) {
                return [row, col];
            }
        }
    }
    throw new Error(`Character '${char}' not found in grid`);
};
```

Rysunek 8 - Funkcje pomocnicze

```
const prepareText = (text) => {  Show usages  taczhed
    text = text.toLowerCase().replace( searchValue: /[^aąbcćdeęfghijklłmnńoópqrsśtuvwxyzźż]/g,  replaceValue: '');
    let result : string  = '';


    for (let i : number  = 0; i < text.length; i++) {
        result += text[i];
        if (i + 1 < text.length && text[i] === text[i + 1]) {
            result += 'x';
        }
    }

    if (result.length % 2 !== 0) {
        result += 'x';
    }
    return result;
};
```

Rysunek 9 - Przygotowanie tekstu



Rysunek 10 - Działanie

6. RSA

Algorytm RSA jest jednym z najpopularniejszych algorytmów szyfrowania asymetrycznego. Wykorzystuje on dwie różne, ale matematycznie powiązane, liczby: klucz publiczny do szyfrowania i klucz prywatny do deszyfrowania.

Funkcje w implementacji:

- gcd(a, b)
  - Opis: Funkcja oblicza największy wspólny dzielnik (NWD) dwóch liczb a i b za pomocą algorytmu Euklidesa.
  - Zwracany wynik: Zwraca największy wspólny dzielnik a i b.
- modInverse(a, m)
  - Opis: Funkcja oblicza odwrotność modularną liczby a względem m, czyli liczbę x
  - Zwracany wynik: Zwraca odwrotność modularną a względem m.
- generateRSAKeys(p, q)
  - Opis: Funkcja generuje parę kluczy RSA: publiczny i prywatny.
  - Przyjmuje jako argumenty dwie liczby pierwsze p i q.
  - Oblicza n = p * q oraz funkcję Eulera phi(n) = (p - 1) * (q - 1).
  - Wybiera e, które jest względnie pierwsze z phi(n).
  - Oblicza odwrotność modularną e względem phi(n) (oznaczoną jako d).
  - Zwracany wynik: Zwraca obiekt zawierający:
    - publicKey: Klucz publiczny { e, n }
    - privateKey: Klucz prywatny { d, n }
- rsaEncrypt(text, publicKey)
  - Opis: Funkcja szyfruje tekst za pomocą klucza publicznego RSA.
  - Tekst jest zamieniany na kody ASCII każdej litery.
  - Zwracany wynik: Zwraca tablicę zaszyfrowanych liczb, reprezentujących tekst po zaszyfrowaniu.
- rsaDecrypt(cipherText, privateKey)
  - Opis: Funkcja odszyfrowuje tekst zaszyfrowany za pomocą klucza prywatnego RSA.
  - Odszyfrowane liczby są konwertowane z powrotem na znaki ASCII.
  - Zwracany wynik: Zwraca odszyfrowany tekst.

Jak to działa:

- Generowanie kluczy:
  - Zaczynamy od wyboru dwóch liczb pierwszych p i q. Następnie obliczamy n oraz funkcję Eulera phi(n). Wybieramy wykładnik publiczny e tak, aby był względnie pierwszy z phi(n). Na końcu obliczamy odwrotność modularną e względem phi(n), co daje nam wykładnik prywatny d.

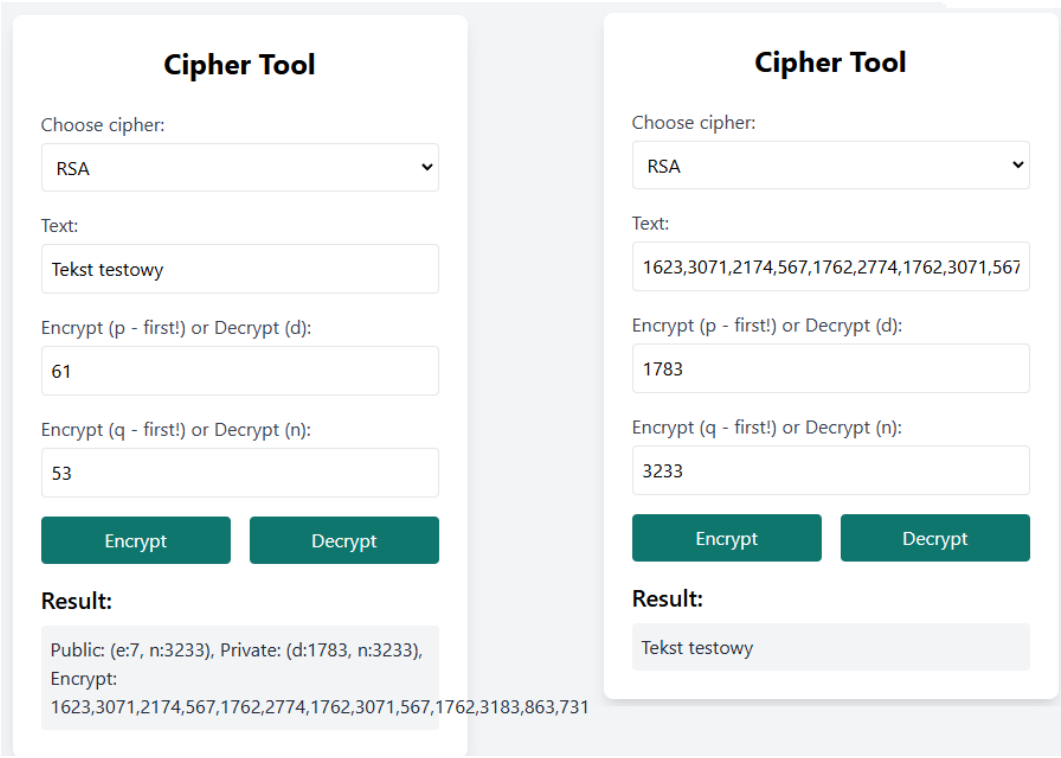
- Szyfrowanie:
    - Aby zaszyfrować wiadomość, przekształcamy każdy znak w jego kod ASCII, a następnie szyfrujemy go za pomocą wzoru RSA
- Odszyfrowywanie:
    - Aby odszyfrować wiadomość, bierze się każdą liczbę (zaszyfrowany znak) i stosuje wzór RSA.
    - Otrzymaną liczbę przekształcamy z powrotem na znak ASCII, co daje nam oryginalny tekst.
- Odszyfrowujemy zaszyfrowaną wiadomość używając klucza prywatnego.

```
export const generateRSAKeys = (p, q) => {  Show usages   taczhed
    let n : number  = p * q;
    let phi : number  = (p - 1) * (q - 1);
    let e : number  = 3;
    while (gcd(e, phi) !== 1) {
        e += 2;
    }
    let d : number  = modInverse(e, phi);

    return {
        publicKey: { e, n },
        privateKey: { d, n }
    };
};

export const rsaEncrypt = (text, publicKey) => {  Show usages   taczhed
    let { e, n } = publicKey;
    return text.split('').map(char => {
        let m : number  = char.charCodeAt(0);
        return BigInt(m) ** BigInt(e) % BigInt(n);
    });
};

export const rsaDecrypt = (cipherText, privateKey) => {  Show usages   taczhed
    let { d, n } = privateKey;
    return cipherText.split(',').map(c => {
        let m : bigint  = BigInt(c) ** BigInt(d) % BigInt(n);
        return String.fromCharCode(Number(m));
    }).join('');
};
```

Rysunek 11 – Kod

```
const gcd = (a, b) => {  Show usages  taczhed
    while (b !== 0) {
        let temp = b;
        b = a % b;
        a = temp;
    }
    return a;
};

const modInverse = (a, m) => {  Show usages  taczhed
    let m0 = m;
    let x0 : number  = 0, x1 : number  = 1;
    if (m === 1) return 0;
    while (a > 1) {
        let q : number  = Math.floor( x: a / m);
        let t : number  = m;
        m = a % m;
        a = t;
        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }
    if (x1 < 0) x1 += m0;
    return x1;
};
```

Rysunek 12 - Funkcje pomocnicze



Rysunek 13 - Działanie