MINISTERUL EDUCATIEI, CULTURII
ȘI CERCETĂRII AL REPUBLICII MOLDOVA
UNIVERSITATEA TEHNICA A MOLDOVEI
Facultatea Calculatoare, Informatică și Microelectronică

Lucrare de laborator nr.1

la disciplina: Securitatea Informațională

Tema: Algoritmi de criptare simetrici (S-DES)

A efectuat: Șaptefrați Victor,
gr. TI-191 F/R

A verificat: Poștaru Andrei
lect. univ.

Chișinău 2024

Introducere

S-DES este o versiune simplificată a algoritmului DES (Data Encryption Standard), proiectat pentru a facilita înțelegerea conceptelor de bază ale criptării. A fost conceput pentru a fi suficient de simplu pentru a fi înțeles și implementat de studenți, dar totodată să reflecte elementele esențiale ale criptării DES.

Principalele caracteristici ale S-DES sunt:
- **Criptarea blocurilor de date**: S-DES criptează și decriptează date în blocuri mici, folosind o cheie de criptare specifică.
- **Chei de criptare**: Utilizează o cheie mai scurtă decât DES standard, ceea ce îl face mai ușor de înțeles și implementat, dar și mai puțin sigur pentru utilizarea în scopuri reale de securitate.
- **Algoritm de substituție-permutație**: Folosește o combinație de tehnici de substituție și permutație, similare cu DES, pentru a realiza criptarea datelor.
- **Runde multiple de procesare**: Execută un număr de runde pentru a transforma datele de intrare în date criptate.

Principalul rol al S-DES este educativ, oferind o introducere în criptografia simetrică și în modul în care funcționează algoritmii de criptare.

Implementare

Implementarea RSA constă din trei etape principale: generarea cheilor, criptarea și decriptarea.

1. Generarea cheii

La S-DES, generarea cheii implică următorii pași tehnici:
- Se alege o cheie de 10 biți.
- Cheia este supusă unei permutări inițiale (P10), care rearanjează biții conform unei scheme prestabilite.
- Cheia permutată este împărțită în două jumătăți de câte 5 biți fiecare.
- Fiecare jumătate este rotită circular (shift left) cu unul sau două poziții, în funcție de rundă.
- Se aplică o a doua permutare (P8) pentru a genera două sub-chei de 8 biți fiecare, folosite în runde diferite de criptare și decriptare.

2. Criptarea

Procesul de criptare în S-DES include următorii pași:
- Datele de intrare (8 biți) sunt permutate inițial folosind o permutare inițială (IP).
- Blocul de date este împărțit în două jumătăți de 4 biți.
- Se aplică două runde de procesare, fiecare folosind o sub-cheie diferită. Fiecare rundă include:
    - Expandarea jumătății drepte la 8 biți și combinarea cu sub-cheia printr-un XOR.
    - Aplicarea unei funcții de substituție (S-box) pentru a reduce din nou la 4 biți.
    - Combinarea rezultatului cu jumătatea stângă prin XOR și schimbarea jumătăților.

- După a doua rundă, jumătățile sunt recombinate și supuse unei permutări finale inverse (IP^-1) pentru a produce blocul criptat.

3. Decriptarea

Decriptarea urmează aceiași pași ca și criptarea, dar în ordine inversă:

- Datele criptate sunt permutate inițial cu IP^-1.
- Se despart în două jumătăți și se procesează prin două runde, folosind sub-cheile în ordine inversă.
- Fiecare rundă include expansiunea, combinarea cu sub-cheia prin XOR, aplicarea S-box-urilor și XOR cu cealaltă jumătate, urmată de schimbarea jumătăților.
- La final, datele sunt recombinate și supuse permutării inițiale IP pentru a obține datele originale.

Codul programului

```
class SDES {
  private key: number[]; // Cheia binară de 10 biți

  private IP: number[]; // Tabel de permutare IP (Initial Permutation)
  private IPInverse: number[]; // Tabel de permutare IP-1 (Initial
Permutation Inverse)

  private P10: number[]; // Tabel de permutare P10
  private P8: number[]; // Tabel de permutare P8
  private P4: number[]; // Tabel de permutare P4

  private BOX_1: number[][]; // S-Box 1 (Matrice 4x4)
  private BOX_2: number[][]; // S-Box 2 (Matrice 4x4)

  private subKeys: { key1: number[]; key2: number[] };

  constructor(key: number[]) {
    this.key = key;

    this.P4 = this.generatePermutationTable(4);
    this.P8 = this.generatePermutationTable(8);
    this.P10 = this.generatePermutationTable(10);

    this.IP = this.getInitialPermutation();
    this.IPInverse = this.getArrayInverse(this.IP);

    this.BOX_1 = this.generateSBox();
    this.BOX_2 = this.generateSBox();

    this.subKeys = this.generateSubKeys();
  }
```

```
  private getInitialPermutation() {
    return new Array(8).fill(0).map((_, i) => i + 1);
  }

  private getArrayInverse(IP: number[]) {
    const IPInverse = new Array(IP.length);
    for (let i = 0; i < IP.length; i++) {
      // The '+1' and '-1' are used to adjust for the fact that array indices
in JavaScript are 0-based
      IPInverse[IP[i] - 1] = i + 1;
    }

    return IPInverse;
  }

  /**
   * Generează un tabel de permutare de lungime specificată.
   * Această metodă poate fi utilizată pentru a genera IP, P10, P8, sau P4.
   *
   * @param length Lungimea tabelului de permutare
   * @return Tabelul de permutare generat
   */
  private generatePermutationTable(length: number): number[] {
    const table = new Array(length);
    for (let i = 0; i < length; i++) {
      table[i] = Math.floor(Math.random() * length) + 1;
    }
    return this.shuffleArray(table);
  }

  /**
   * Amestecă aleatoriu un array.
   *
   * @param array Array-ul de amestecat
   * @return Array-ul amestecat
   */
  private shuffleArray(array: number[]): number[] {
    for (let i = array.length - 1; i > 0; i--) {
      const j = Math.floor(Math.random() * (i + 1));
      [array[i], array[j]] = [array[j], array[i]];
    }
    return array;
  }

  /**
   * Generează un S-Box de dimensiuni 4x4.
   *
```

```
   * @return S-Box-ul generat
   */
  private generateSBox(): number[][] {
    const sBox = new Array(4);
    for (let i = 0; i < 4; i++) {
      sBox[i] = this.shuffleArray([0, 1, 2, 3]);
    }
    return sBox;
  }

  private permute(input: number[], permutationTable: number[]): number[] {
    return permutationTable.map((index) => input[index - 1]);
  }

  private circularLeftShift(input: number[], shiftNumber: number): number[] {
    return input.slice(shiftNumber).concat(input.slice(0, shiftNumber));
  }

  /**
   * Primul pas în S-DES este generarea a două subchei de 8 biți fiecare
dintr-o cheie de 10 biți.
   * Să începem cu implementarea acestei logici.
   *
   * - *Permutare*: Aplicăm o permutare asupra cheii inițiale de 10 biți.
   * - *Shiftare circulară*: Aplicăm un shift circular la stânga pe fiecare
jumătate a cheii permutate.
   * - *Generarea Subcheii 1*: Formăm prima subcheie folosind o a doua
permutare.
   * - *Shiftare circulară (din nou)*: Aplicăm încă un shift circular la
stânga pe ambele jumătăți.
   * - *Generarea Subcheii 2*: Formăm a doua subcheie.
   */
  private generateSubKeys(): { key1: number[]; key2: number[] } {
    // Aplică permutarea P10
    const permutedKey = this.permute(this.key, this.P10);

    // Împarte cheia permutată în două jumătăți și aplică shift circular
    let left = permutedKey.slice(0, 5);
    let right = permutedKey.slice(5, 10);
    left = this.circularLeftShift(left, 1);
    right = this.circularLeftShift(right, 1);

    // Formează prima subcheie
    const key1 = this.permute(left.concat(right), this.P8);

    // Aplică un alt shift circular pentru a forma a doua subcheie
    left = this.circularLeftShift(left, 2);
    right = this.circularLeftShift(right, 2);
```

```
    const key2 = this.permute(left.concat(right), this.P8);

    return { key1, key2 };
  }

  /**
   * Procesul de criptare în S-DES implică mai mulți pași:
   *
   * Permutarea inițială (IP): Aplicăm o permutare inițială pe textul clar.
   * Rundele de procesare: Executăm două runde de procesare, folosind fiecare
subcheie.
   * Permutarea finală (IP-1): Aplicăm o permutare inversă pentru a obține
textul criptat.
   */
  public encrypt(value: number[]): number[] {
    // Aplică permutarea inițială
    let data = this.permute(value, this.IP);

    // Împarte datele în două jumătăți
    let left = data.slice(0, 4);
    let right = data.slice(4, 8);

    // Runda 1
    [left, right] = this.round(left, right, this.subKeys.key1);

    // Schimbă jumătățile
    [left, right] = [right, left];

    // Runda 2
    [left, right] = this.round(left, right, this.subKeys.key2);

    // Aplică permutarea finală
    return this.permute(left.concat(right), this.IPInverse);
  }

  // Mărim și rearanjăm jumătatea dreaptă a datelor.
  private expandAndPermute(right: number[]): number[] {
    const expansionPermutation = [4, 1, 2, 3, 2, 3, 4, 1];
    return this.permute(right, expansionPermutation);
  }

  // Aplicăm XOR între datele expandate și subcheie.
  private xor(bits1: number[], bits2: number[]): number[] {
    return bits1.map((bit, i) => bit ^ bits2[i]);
  }

  /**
```

```
   * Transformăm datele folosind S-Boxes, care sunt tabele predefinite de
valori.
   *
   * Pentru a aplica S-Box, împărțim datele în două jumătăți și aplicăm S-Box
pe fiecare jumătate.
   *
   * A aplica S-Box pe o jumătate implică:
   * 1. Determinăm rândul și coloana din S-Box folosind primul și ultimul bit
din jumătatea curentă.
   * 2. Determinăm valoarea din S-Box folosind rândul și coloana.
   * 3. Convertim valoarea în binar și o returnăm.
   */
  private sBox(input: number[]): number[] {
    // Imparte input-ul în două și aplică S-Boxes
    const leftInput = input.slice(0, 4);
    const rightInput = input.slice(4, 8);

    const row1 = (leftInput[0] << 1) + leftInput[3];
    const col1 = (leftInput[1] << 1) + leftInput[2];
    const row2 = (rightInput[0] << 1) + rightInput[3];
    const col2 = (rightInput[1] << 1) + rightInput[2];

    // Converteste valorile S-Box în binar
    const sBox1Result = this.BOX_1[row1][col1]
      .toString(2)
      .padStart(2, "0")
      .split("")
      .map(Number);
    const sBox2Result = this.BOX_2[row2][col2]
      .toString(2)
      .padStart(2, "0")
      .split("")
      .map(Number);

    return sBox1Result.concat(sBox2Result);
  }

  private p4Permutation(input: number[]): number[] {
    return input.map((_, i) => input[this.P4[i] - 1]);
  }

  /**
   * Rundele reprezintă cea mai importantă parte a algoritmului S-DES.
   *
   * Fiecare rundă implică mai mulți pași:
   * - *Expandare și permutare*: Mărește și rearanjează jumătatea dreaptă a
datelor.
   * - *XOR*: Aplică XOR între datele expandate și subcheie.
```

```
   * - *S-Box*: Transformă datele folosind S-Boxes, care sunt tabele
predefinite de valori.
   * - *P4*: Aplică o permutare P4.
   * - *XOR*: Aplică XOR între jumătatea stângă și rezultatul P4.
   *
   * @param left
   * @param right
   * @param key
   * @returns
   */
  private round(
    left: number[],
    right: number[],
    key: number[]
  ): [number[], number[]] {
    // Expandează și permută jumătatea dreaptă
    const expandedRight = this.expandAndPermute(right);

    // Aplică XOR cu subcheia
    const xorResult = this.xor(expandedRight, key);

    // Aplică S-Box și P4 permutare
    const sBoxResult = this.sBox(xorResult);
    const p4Result = this.p4Permutation(sBoxResult);

    // Aplică XOR cu jumătatea stângă și întoarce rezultatul
    return [this.xor(left, p4Result), right];
  }

  public decrypt(ciphertext: number[]): number[] {
    // Aplică permutarea inițială
    let data = this.permute(ciphertext, this.IP);
    // Împarte datele în două jumătăți
    let left = data.slice(0, 4);
    let right = data.slice(4, 8);

    // Runda 1 cu key2
    [left, right] = this.round(left, right, this.subKeys.key2);

    // Schimbă jumătățile
    [left, right] = [right, left];

    // Runda 2 cu key1
    [left, right] = this.round(left, right, this.subKeys.key1);

    // Aplică permutarea finală
    return this.permute(left.concat(right), this.IPInverse);
  }
```

```
  public encryptText(plaintext: string): string {
    const binaryPlaintext = this.stringToBinary(plaintext);

    let encryptedBinary: number[] = [];
    for (let i = 0; i < binaryPlaintext.length; i += 8) {
      let block = binaryPlaintext.slice(i, i + 8);
      encryptedBinary.push(...this.encrypt(block));
    }

    return this.binaryToString(encryptedBinary);
  }

  public decryptText(ciphertext: string): string {
    const binaryEncryptedText = this.stringToBinary(ciphertext);

    let decryptedBinary: number[] = [];
    for (let i = 0; i < binaryEncryptedText.length; i += 8) {
      let block = binaryEncryptedText.slice(i, i + 8);
      decryptedBinary.push(...this.decrypt(block));
    }
    return this.binaryToString(decryptedBinary);
  }

  private stringToBinary(input: string): number[] {
    return input
      .split("")
      .map((char) => char.charCodeAt(0).toString(2).padStart(8, "0")) //
Convert to binary and pad
      .join("")
      .split("")
      .map((num) => parseInt(num));
  }

  private binaryToString(input: number[]): string {
    let binaryString = input.join("");
    let output = "";
    for (let i = 0; i < binaryString.length; i += 8) {
      let byte = binaryString.slice(i, i + 8);
      output += String.fromCharCode(parseInt(byte, 2));
    }
    return output;
  }
}
```

Execuția programului

```
const key = [0,0,0,1,0,1,0,0,1,1];
// sau putem genera o cheie aleatoare:
// new Array(10).fill(0).map(() => Math.round(Math.random()));
console.log("Key:", JSON.stringify(key));

const sdes = new SDES(key);
const plaintext = "S-DES@vixeven";
const encrypted = sdes.encryptText(plaintext);
const decrypted = sdes.decryptText(encrypted);

console.log("Encrypted:", encrypted);
console.log("Decrypted:", decrypted);
```

```
Key:       [0,0,0,1,0,1,0,0,1,1]
Encrypted: +m+´³]
Decrypted: S-DES@vixeven
```