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

Lucrare de laborator nr.3

la disciplina: Securitatea Informațională

Tema: Semnătura Digitală (DSA)

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

A verificat: Poștaru Andrei
lect. univ.

Chișinău 2024

Introducere

DSA (Digital Signature Algorithm) este un standard pentru semnături digitale, care permite unei părți să demonstreze autenticitatea unui mesaj electronic. A fost introdus în anul 1991 de către National Institute of Standards and Technology (NIST) ca parte a standardului de securitate digitală FIPS 186, destinat pentru a asigura autenticitatea și integritatea documentelor digitale.

Acesta oferă două capabilități cheie: semnarea unui mesaj și verificarea acestei semnături. Este bazat pe algoritmi de matematică modulară și logaritmi discreți.

Funcțiile de bază a acestui algoritm sunt:
- *Autenticitate*: DSA permite verificarea că un mesaj a fost semnat de un anumit emițător, oferind astfel autenticitate.
- *Integritate*: Garantează că mesajul nu a fost modificat de la semnare, asigurând integritatea datelor.
- *Non-repudiere*: Semnătura digitală previne emitentul să nege trimiterea sau semnarea mesajului.

Implementare

Implementarea DSA constă din mai trei etape principale: generarea cheilor, semnarea și verificarea.

1. Generarea cheilor

Pentru început se vor alege parametrii, după următoarele reguli:
- Se alege un număr prim mare q.
- Se alege un alt număr prim, astfel încât p – 1 să fie divizibil cu q.
- Se alege un număr g, un generator al grupului multiplicativ al resturilor modulului p, astfel încât ordinul său să fie q.

Din moment ce avem aceste variabile putem să generăm cheia privată:
- Se alege un număr aliator x, astfel încât $0 < x < q$, unde x este cheia privata.

Ultimul pas este să generăm cheia publică, care este calculată după următoarea formulă:
- Se calculează $y = g^x \ mod \ p$, unde y este cheia publică.

2. Crearea semnăturii

Pentru fiecare semnătură se va alege un număr aliator k, astfel încât $0 < k < q$.

Semnătura unui mesaj este reprezentată de o pereche de numere, r și s, care sunt calculate după următoarele formule:

$$r = (g^k \ mod \ p) \ mod \ q$$

$$s = k^{-1}(H(m) + x * r) \ mod \ q,$$

unde $H(m)$ este hash-ul mesajului care trebuie semnat.

3. Validarea semnăturii

Se calculează $w = s^1 \ mod \ q$, iar apoi $v = \ (g^{u1} * y^{u2}) \ mod \ p) \ mod \ q$,
unde $u_1 = H(m) * w \ mod \ q$, iar $u_2 = r * w \ mod \ q$.

Dacă la final expresia matematică $v = r$ este adevărată, semnătura este validă.

Codul programului

```
class DSA {
  private static p: bigint;
  private static q: bigint;
  private static g: bigint;
  private static x: bigint;
  private static y: bigint;

  public static generateKeys() {
    // Prentru simplitate, am ales valori fixe pentru `p` și `q`.
    this.p = BigInt(100003263001);

    // In practica, `q` ar trebui sa fie ales cu grija,
    // astfel incat `p - 1` sa fie divizibil cu `q`.
    this.q = BigInt(709243);

    // Generăm `x` și `g` în mod aleatoriu.
    this.x = this.generateX(); // `0 < x < q`
    this.g = this.generateG(); // `g^q mod p = 1`

    // Calculăm `y` folosind formula: `y = g^x mod p`.
    this.y = this.modPow(this.g, this.x, this.p);

    return {
      p: this.p,
      q: this.q,
      g: this.g,
      x: this.x,
      y: this.y,
    };
  }

  /**
   * Generăm un număr aleatoriu `x` astfel încât `0 < x < q`.
   */
  private static generateX() {
    let x: number;

    do {
```

```
      x = Math.floor(Math.random() * Number(this.q));
    } while (x <= 0n || x >= this.q);

    return BigInt(x);
  }

  /**
   * Generăm un număr aleatoriu `g` astfel încât `g^q mod p = 1`.
   */
  private static generateG() {
    let g: number;

    while (true) {
      g = Math.floor(Math.random() * Number(this.p));

      if (this.modPow(BigInt(g), this.q, this.p) === 1n) {
        break;
      }
    }

    return BigInt(g);
  }

  /**
   * Generăm un număr aleatoriu `k` astfel încât `0 < k < q`.
   *
   * Acesta este folosit pentru a calcula `r` și `s`,
   * care sunt folosite pentru a forma semnătura.
   */
  private static generateK() {
    let k: number;
    do {
      k = Math.floor(Math.random() * Number(this.q));
    } while (k <= 0n || k >= this.q);

    return BigInt(k);
  }

  /**
   * Functia care formeaza semnatura.
   *
   * Aceasta returneaza un obiect cu două proprietăți:
   * `r` — este calculat folosind formula: `r = (g^k mod p) mod q`,
   * `s` — este calculat folosind formula: `s = k^-1 (H(m) + x * r) mod q`.
   *
   * @param message Mesajul care trebuie semnat
   * @returns Semnatura { r, s }
   */
```

```
  public static sign(message: string) {
    const hash = this.hash(message);
    const k = this.generateK();
    const r = this.modPow(this.g, k, this.p) % this.q;
    const s = (this.modInverse(k, this.q) * (hash + this.x * r)) % this.q;

    if (s === 0n) {
      throw new Error("Semnatura invalida. `s` nu poate fi 0.");
    }

    if (r === 0n) {
      throw new Error("Semnatura invalida. `r` nu poate fi 0.");
    }

    return { r, s };
  }

  /**
   * Verifică dacă semnătura este validă.
   *
   * @param message Mesajul care trebuie verificat
   * @param signature Semnătura cu care va fi verificat mesajul
   *
   * @returns `true` dacă semnătura este validă, `false` în caz contrar
   */
  public static verify(message: string, { r, s }: { r: bigint; s: bigint }) {
    // Calculează hash-ul mesajului.
    // În DSA, este comun să se folosească o funcție de hash pentru a reduce
mesajul la o valoare fixă.
    const hash = this.hash(message);

    // Calculează inversul modular al lui 's' în raport cu 'q'.
    // Aceasta este parte din procesul de verificare și este folosit în
calculele ulterioare.
    const w = this.modInverse(s, this.q);

    // Calculează u1 și u2 folosind hash-ul, 'r', 's', și 'q'.
    // Acești parametri sunt folosiți pentru a reconstrui semnătura.
    const u1 = this.modPow(hash * w, 1n, this.q); // H(m) * w mod q
    const u2 = this.modPow(r * w, 1n, this.q); // r * w mod q

    // Calculează 'v', care este valoarea verificată împotriva lui 'r' pentru
a determina validitatea semnăturii.
    // Acest calcul se face folosind formula: `v = ((g^u1 * y^u2 mod p) mod
q) mod q`.
    const v =
      ((this.modPow(this.g, u1, this.p) * this.modPow(this.y, u2, this.p)) %
        this.p) %
```

```
      this.q;

    return v === r;
  }

  /**
   * Calculează hash-ul mesajului folosind SHA-256.
   *
   * @param message Mesajul care urmează să fie hash-uit.
   *
   * @returns Hash-ul mesajului, reprezentat ca număr.
   */
  private static hash(message: string) {
    const hash = createHash("sha256").update(message).digest("hex");

    return BigInt("0x" + hash);
  }

  /**
   * Calculează `base^exponent mod modulus` folosind exponentierea modulară.
   *
   * Este alternativa pentru expresia `Math.pow(base, exponent) % modulus`,
   * care permite calcularea rezultatului fără a depăși limitele numerice.
   */
  private static modPow(base: bigint, exponent: bigint, modulus: bigint) {
    let result = BigInt(1);
    base = base % modulus;

    while (exponent > 0) {
      if (exponent % BigInt(2) === BigInt(1)) {
        result = (result * base) % modulus;
      }
      exponent = exponent >> BigInt(1);
      base = (base * base) % modulus;
    }

    return result;
  }

  /**
   * Calculează inversul modular `d ≡ e^(-1) mod φ`,
   * folosind algoritmul extins al lui Euclid.
   *
   * @param e Exponentul public, reprezentat ca un bigint.
   * @param phi φ, totientul folosit pentru a calcula cheile, reprezentat ca
un bigint.
   * @returns Inversul modular al lui e modulo φ.
   */
```

```
  private static modInverse(e: bigint, phi: bigint) {
    // Salvăm valoarea inițială a lui φ pentru a ajusta rezultatul final dacă
este negativ.
    let m0 = phi;

    // Inițializăm y și x, care sunt folosite în algoritmul extins al lui
Euclid.
    // y va fi folosit pentru a stoca valorile intermediare ale inversului
modular,
    // în timp ce x va stoca valoarea anterioară a lui y în fiecare iterație.
    let y = BigInt(0);
    let x = BigInt(1);

    // Dacă φ este 1, inversul modular nu există, deci returnăm 0.
    if (phi === BigInt(1)) return BigInt(0);

    // Executăm algoritmul extins al lui Euclid.
    // Acesta funcționează prin găsirea coeficienților x și y astfel încât e
* x + φ * y = gcd(e, φ).
    while (e > 1) {
      // Calculăm partea întreagă a împărțirii lui e la φ.
      let q = e / phi;

      // Actualizăm valorile lui phi și e folosind algoritmul de schimbare
Euclidian.
      let t = phi;
      phi = e % phi;
      e = t;

      t = y; // Actualizăm valorile lui y și x.
      y = x - q * y; // Calculează noua valoare a lui y.
      x = t; // Actualizează x la valoarea anterioară a lui y.
    }

    // Dacă x este negativ, adăugăm m0 (valoarea originală a lui phi) pentru
a-l face pozitiv.
    // Inversul modular trebuie să fie întotdeauna un număr pozitiv.
    if (x < 0) x += m0;

    // Returnăm inversul modular calculat.
    return x;
  }
}
```

Execuția programului

```
const message = "UTM 2023!";

const { x, y } = DSA.generateKeys();
console.log("Public key:", y);
console.log("Private key:", x);

const signature = DSA.sign(message);
console.log("Signature:", signature);

const isValid = DSA.verify(message, signature);
console.log("Signature is valid:", isValid);
```

```
Public key: 25535036958n
Private key: 312436n
Signature: { r: 87461n, s: 694944n }
Signature is valid: true
```