МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТИХНІЧНИЙ ІНСТИТУТ ІМ.ІГОРЯ СІКОРСЬКОГО»

ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Кафедра інформаційної безпеки

Комп’ютерний практикум №4

«Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем»

Виконали:

Студенти 3 курсу

Групи ФБ-94

Волков Артем та Калінічев Сергій

Київ

НТУУ «КПІ»

Мета та основні завдання роботи

Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

Завдання 1

```rust
fn miller_rabin(p: BigInt, k: usize) -> bool {
    let mut rng : ThreadRng = rand::thread_rng();
    let zero : BigInt = 0.to_bigint().unwrap();
    let one : BigInt = 1.to_bigint().unwrap();
    let two : BigInt = 2.to_bigint().unwrap();

    let p_minus_one : BigInt = &p - &one;
    let mut d : BigInt = p_minus_one.clone();
    let mut s : BigInt = zero.clone();

    while &d % &two == one {
        d /= &two;
        s += &one;
    }

    'l: for _ in 0..k {
        let x : BigInt = rng.gen_bigint_range( lbound: &two, ubound: &d);
        let mut v : BigInt = x.modpow( exponent: &d, modulus: &p);
        if &v == &one {
            continue 'l;
        }
        let mut i : BigInt = zero.clone();
        while &i < &s {
            v = (v.clone() * &v) % &p;
            if &v == &p_minus_one {
                continue 'l;
            }
            i += &one;
        }
        return false;
    }
    true
}
```

Завдання 2

```rust
fn gen_prime() -> BigUint {
    let mut rng : ThreadRng = rand::thread_rng();
    loop {
        let signed: BigUint = rng.sample( distr: RandomBits::new( bits: 256));
        if miller_rabin( p: signed.to_bigint().unwrap(), k: 100) {
            return signed;
        }
    }
}

fn gen_pair() -> (BigUint, BigUint) {
    (gen_prime(), gen_prime())
}

fn gen_compatible_pair(p: BigUint, q: BigUint) -> (BigUint, BigUint) {
    let n : BigUint = p * q;
    loop {
        let (p : BigUint, q : BigUint ) = gen_pair();
        if n <= &p * &q {
            return (p, q);
        }
    }
}
```

Завдання 3

```rust
#[derive(Debug)]
struct User {
    name: String,
    n: BigUint,
    exp: BigUint,
    d: BigUint,
    received_exp: Option<BigUint>,
    received_n: Option<BigUint>,
}
```

```rust
impl User {
    fn new(name: String, p: BigUint, q: BigUint) -> Self {
        let one : BigUint = 1.to_biguint().unwrap();
        let totient : BigUint = (&p - &one) * (&q - &one);
        let mut rng : ThreadRng = rand::thread_rng();
        loop {
            let exp : BigUint = rng.gen_biguint_range( lbound: &2.to_biguint().unwrap(), ubound: &totient);
            if &exp.gcd( other: &totient) == &one {
                println!("Create user: {}", &name);
                println!("p: {}\nq: {}", &p, &q);
                println!("exp: {}", &exp);
                let n : BigUint = p * q;
                let d : BigUint = exp
                    .clone() : BigUint
                    .mod_inverse( m: &totient) : Option<BigInt>
                    .unwrap() : BigInt
                    .to_biguint() : Option<BigUint>
                    .unwrap();
                println!("n: {}", &n);
                println!("d: {}", &d);
                return User {
                    name,
                    n,
                    exp,
                    d,
                    received_exp: None,
                    received_n: None,
                };
            }
        }
    }
}
```

```rust
let mut alice : User = User::new( name: "Alice".to_string(), p, q);
let mut bob : User = User::new( name: "Bob".to_string(), p: p1, q: p2);

bob.receive_key( key: alice.send_key());
alice.receive_key( key: bob.send_key());
```

Завдання 4-5

```rust
fn send_message(&self, text: String) -> String {
    let m : BigUint = BigUint::from_bytes_be( bytes: text.as_bytes());
    let encrypted : String = m
        .modpow(
            exponent: &self.received_exp.as_ref().unwrap(),
            modulus: &self.received_n.as_ref().unwrap(),
        ) : BigUint
        .to_str_radix( radix: 16u32);
    println!(
        "{} send message: {}\nencrypted: {}\n",
        &self.name, text, &encrypted
    );
    encrypted
}

fn receive_message(&self, text: String) -> String {
    let c : BigUint = BigUint::from_str_radix( s: &text, radix: 16u32).unwrap();
    let bytes : Vec<u8> = c.modpow( exponent: &self.d, modulus: &self.n).to_bytes_be();
    let decrypted : String = String::from_utf8( vec: bytes).unwrap();
    println!(
        "{} receive message: {}\ndecrypted: {}\n",
        &self.name, text, &decrypted
    );
    decrypted
}
```

```rust
fn sing_message(&self, text: String) -> (String, String) {
    let m : BigUint = BigUint::from_bytes_be( bytes: &text.as_bytes());
    let signature : String = m.modpow( exponent: &self.d, modulus: &self.n).to_str_radix( radix: 16u32);
    println!(
        "{} sing message: {}\nsignature: {}\n",
        &self.name, text, &signature
    );
    (signature, text)
}

fn verify_signature(&self, signature: String, message: String) -> bool {
    let m : BigUint = BigUint::from_str_radix( s: &signature, radix: 16u32).unwrap();
    let m : Vec<u8> = m
        .modpow(
            exponent: &self.received_exp.as_ref().unwrap(),
            modulus: &self.received_n.as_ref().unwrap(),
        ) : BigUint
        .to_bytes_be();
    let message1 : String = String::from_utf8( vec: m).unwrap();
    if message == message1 {
        println!(
            "{} successfully verify signature {} for message: {}",
            &self.name, &signature, &message
        );
        true
    } else {
        false
    }
}
```

```rust
fn send_key(&self) -> (BigUint, BigUint) {
    println!("{} send key", &self.name);
    (self.exp.clone(), self.n.clone())
}

fn receive_key(&mut self, key: (BigUint, BigUint)) {
    println!("{} receive key", &self.name);
    self.received_exp = Some(key.0);
    self.received_n = Some(key.1);
}
```

```rust
fn main() {
    let (p : BigUint , q : BigUint ) = gen_pair();
    let (p1 : BigUint , p2 : BigUint ) = gen_compatible_pair( p: p.clone(), q: q.clone());

    let mut alice : User = User::new( name: "Alice".to_string(), p, q);
    let mut bob : User = User::new( name: "Bob".to_string(), p: p1, q: p2);

    bob.receive_key( key: alice.send_key());
    alice.receive_key( key: bob.send_key());

    let encrypted : String = alice.send_message( text: "Hello Bob!".to_string());
    bob.receive_message( text: encrypted);

    let (signature : String , message : String ) = alice.sing_message( text: "Bob check my sign".to_string());
    bob.verify_signature(signature, message);
}
```

```
C:/Users/admin/.cargo/bin/cargo.exe run --color=always --package volkov_fb-94_kalinichev_fb-94_cp4 --bin volkov_fb-94_kalinichev_fb-94_cp4
    Finished dev [unoptimized + debuginfo] target(s) in 0.10s
     Running `target\debug\volkov_fb-94_kalinichev_fb-94_cp4.exe`
Create user: Alice
p: 99397679878808205673286632802619808071490368727112075903753599957852271958181
q: 54300321048620910246362632639936745830307971571205070322742394187417829820991
exp: 3091211190714106432446392453716727438670153460063841174159438822823582343576818000882749957300000082186050054331862894243355226192496746013325195272981447
n: 5397325928907332337577704054738602374534973947333513443673266039813104086647738645092268424224672922905943668012405468003539491126139921430956416267977371
d: 5058618430877431665853197821780707731704890140947396831985868986666333449638272025395532112581903202697880453442568413022791804790439762367580906094867783
Create user: Bob
p: 10023047356132453466458532214521037484063874188147362271376588453884702234733 3
q: 954088085169850327026758358850247632412222705238947357456550496777000784559499
exp: 663857657604587839838939627748590118675224234169855553326839701673054396109811529667919840005201779372786793774124583619124591008866162832466786797198138 89
n: 956287005957914340540881570698186904334119743813819703388047348403796993747400248558146882512838222406824091366147109268293768228190872687704646565382466167
d: 198399476602852334465576805046283699932888303333726290761866050686483033250851429086827266860780712413831847389868878588696784477095771688989842191512230 5
Alice send key
Bob receive key
Bob send key
Alice receive key
Alice send message: Hello Bob!
encrypted_text: 3fdc0ad317e8a0f334ba4f2c7f9f3716241cb6ca4845c597a1d1ccf91ce4e9b9512580b487513cf13ad8f58223305f82e4c31040a7e3fc881235b3d2ddce65f4
encrypted_integer: 33445955042746665375149860814555866532335702243431117558494561517638411317807477941535423599023464777398821327740193693342873373913984582797371351172643 72

Bob receive message: 3fdc0ad317e8a0f334ba4f2c7f9f3716241cb6ca4845c597a1d1ccf91ce4e9b9512580b487513cf13ad8f58223305f82e4c31040a7e3fc881235b3d2ddce65f4
decrypted_text: Hello Bob!
decrypted_integer: 341881320659697045299745
Alice sing message: Bob check my sign
signature: 4d21144b64f7288045e62d0b02a61d4a28fcdf4b0733ad29b98874f923c538f7ed18705fcac607fc3889ccc49f87dc2e9e3174c930f76126ea1d2834fb771214

Bob successfully verify signature 4d21144b64f7288045e62d0b02a61d4a28fcdf4b0733ad29b98874f923c538f7ed18705fcac607fc3889ccc49f87dc2e9e3174c930f76126ea1d2834fb771214 for message: Bob check my sign

Process finished with exit code 0
```

Висновок: Завдяки цьому комп’ютерному практикуму нами були здобуті таки навички: перевірка великих чисел на простоту, генерація ключів для RSA, організація засекреченого каналу зв’язку з подальшим обміном повідомленнями та верифікацією ЕЦП.