`

Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине «Криптографические методы защиты информации»

Тема: «Алгоритмы обмена ключами»

Выполнил:

Студент 3 курса

Группы ИИ-21

Кирилович А. А.

Проверил:

Хацкевич А. С.

Брест 2023

`

Цель: изучить алгоритмы обмена ключами. Практически реализовать алгоритмы обмена ключами.

Ход работы:

Вариант:Расширенный EKE , RC-4

Код сервера:

```
mod eke;
use eke::{generate_prime, generate_private_key, compute_public_key, compute_shared_secret};

use num_bigint::BigUint;
use std::io::{Read, Write};
use std::net::{TcpListener, TcpStream};
use std::thread;
use byteorder::{BigEndian, ReadBytesExt, WriteBytesExt};
use rc4::Cipher;

fn main() {
    let listener = TcpListener::bind("127.0.0.1:8081").expect("Failed to bind");

    for stream in listener.incoming() {
        match stream {
            Ok(mut client) => {
                let prime = generate_prime(512);
                let generator = BigUint::from(2u32);

                send_biguint(&mut client, &prime);
                send_biguint(&mut client, &generator);
                let client_public_key = receive_biguint(&mut client);
                let server_private_key = generate_private_key(&prime);
                let server_public_key = compute_public_key(&generator, &prime, &server_private_key);
                send_biguint(&mut client, &server_public_key);
                let shared_secret = compute_shared_secret(&client_public_key, &prime, &server_private_key);
                println!("Shared secret: {}", shared_secret);
                let mut rc4 = rc4::Cipher::new(&shared_secret.to_bytes_be()).unwrap();
                thread::spawn(move || {
                    handle_client(&mut rc4, &mut client);
                });
            }
            Err(e) => {
                eprintln!("Error accepting connection: {}", e);
            }
        }
    }
}

fn handle_client(rc4: &mut Cipher, client: &mut TcpStream) {
    let mut buffer = [0; 1024];

    loop {
        match client.read(&mut buffer) {
            Ok(0) => {
                println!("The client closed the connection");
                break;
            }
            Ok(bytes_read) => {
                    let mut dst: Vec<u8> = vec![0; bytes_read];
                    rc4.xor(&buffer[0..bytes_read], &mut dst);
                    if let Ok(string_result) = std::str::from_utf8(&dst) {
                        println!("Received from client: {}", string_result);
                    } else {
                        println!("Conversion to String failed");
                    }
            }
            Err(e) => {
                eprintln!("Error reading data: {:?}", e);
                break;
            }
        }
    }
}

fn send_biguint(stream: &mut dyn Write, num: &BigUint) {
    let num_bytes = num.to_bytes_be();
    stream.write_u64::<BigEndian>(num_bytes.len() as u64).unwrap();
    stream.write_all(&num_bytes).unwrap();
}

fn receive_biguint(stream: &mut dyn Read) -> BigUint {
    let size = stream.read_u64::<BigEndian>().unwrap() as usize;
    let mut num_bytes = vec![0; size];
    stream.read_exact(&mut num_bytes).unwrap();
    let num = BigUint::from_bytes_be(&num_bytes);
    num
}
```

`

Код клиента:

```
mod eke;
use eke::{generate_private_key, compute_public_key, compute_shared_secret};


use num_bigint::BigUint;
use std::io::{self, BufRead, Write, Read};
use std::net::TcpStream;
use byteorder::{BigEndian, ReadBytesExt, WriteBytesExt};
use rc4::Cipher;


fn main() {
    match TcpStream::connect("127.0.0.1:8081") {
        Ok(mut server) => {
            let prime = receive_biguint(&mut server);
            let generator = receive_biguint(&mut server);
            let client_private_key = generate_private_key(&prime);
            let client_public_key = compute_public_key(&generator, &prime, &client_private_key);
            send_biguint(&mut server, &client_public_key);
            let server_public_key = receive_biguint(&mut server);
            let shared_secret = compute_shared_secret(&server_public_key, &prime, &client_private_key);
            println!("Shared secret: {}", shared_secret);
            let mut rc4 = rc4::Cipher::new(&shared_secret.to_bytes_be()).unwrap();
            handle_server(&mut rc4, &mut server);
        }
        Err(e) => {
            eprintln!("Error connecting to server: {}", e);
        }
    }
}


fn handle_server(rc4: &mut Cipher, server: &mut TcpStream) {
    let stdin = io::stdin();
    let mut reader = stdin.lock();
    let mut buffer = String::new();
    loop {
        reader.read_line(&mut buffer).expect("Error reading line from console");
        let mut dst: Vec<u8> = vec![0; buffer.len()];
        rc4.xor(buffer.as_bytes(), &mut dst);
        server.write_all(&dst).expect("Error sending data");
        buffer.clear();
    }
}


fn send_biguint(stream: &mut dyn Write, num: &BigUint) {
    let num_bytes = num.to_bytes_be();
    stream.write_u64::<BigEndian>(num_bytes.len() as u64).unwrap();
    stream.write_all(&num_bytes).unwrap();
}


fn receive_biguint(stream: &mut dyn Read) -> BigUint {
    let size = stream.read_u64::<BigEndian>().unwrap() as usize;
    let mut num_bytes = vec![0; size];
    stream.read_exact(&mut num_bytes).unwrap();
    let num = BigUint::from_bytes_be(&num_bytes);
    num
}
```

EKE:

```
use num_bigint::BigUint;
use num_traits::{One, Zero};
use rand::Rng;
use num_integer::Integer;


// Random Prime number generation
pub fn generate_prime(bit_size: usize) -> BigUint {
    let mut rng = rand::thread_rng();
    loop {
        let prime_candidate = generate_random(bit_size);
        if is_prime(&prime_candidate) {
            return prime_candidate;
        }
    }
}


// Random number generation dimension bit_size bit
fn generate_random(bit_size: usize) -> BigUint {
    let mut rng = rand::thread_rng();
    let random_bytes: Vec<u8> = (0..(bit_size + 7) / 8)
        .map(|_| rng.gen())
        .collect();
    BigUint::from_bytes_be(&random_bytes)
}


// Verify prime number with Miller-Rabin test
fn is_prime(n: &BigUint) -> bool {
```

```
    `

    if n == &BigUint::zero() || n == &BigUint::one() {
        return false;
    }
    let mut rng = rand::thread_rng();
    let num_trials = 10;
    for _ in 0..num_trials {
        let a = BigUint::from(2u32 + rng.gen_range(0..100));
        if !miller_rabin(n, &a) {
            return false;
        }
    }
    true
}

// Miller-Rabin test implementation prime number
fn miller_rabin(n: &BigUint, a: &BigUint) -> bool {
    let one = BigUint::one();
    let d = n - &one;
    let mut s = 0;
    let mut d = d.clone();
    while d.is_even() {
        d >>= 1;
        s += 1;
    }

    let mut x = a.modpow(&d, n);
    if x == one || x == n - &one {
        return true;
    }

    for _ in 0..s - 1 {
        x = x.modpow(&2u32.into(), n);
        if x == one {
            return false;
        }
        if x == n - &one {
            return true;
        }
    }

    false
}

// Private key random generation
pub fn generate_private_key(prime: &BigUint) -> BigUint {
    let mut rng = rand::thread_rng();
    loop {
        let private_key = generate_random(prime.bits().try_into().unwrap());
        if private_key > BigUint::zero() && private_key < *prime {
            return private_key;
        }
    }
}

// Public key computation from private key
pub fn compute_public_key(generator: &BigUint, prime: &BigUint, private_key: &BigUint) -> BigUint {
    generator.modpow(private_key, prime)
}

// Secret shared key generation from public and private key
pub fn compute_shared_secret(public_key: &BigUint, prime: &BigUint, private_key: &BigUint) -> BigUint {
    public_key.modpow(private_key, prime)
}
```

Вывод: в ходе лабораторной работы я изучил алгоритмы обмена ключами, а также реализовал клиент-серверное приложение, позволяющее обмениваться зашифрованными сообщениями.