SỐ NGUYÊN TỐ & TRAO ĐỔI KHÓA DIFFIE-HELLMAN

Bộ môn Công nghệ Tri thức

Mục tiêu

Trong bài tập này, sinh viên sẽ hoàn thành các công việc cơ bản sau:

- Triển khai một hàm để thực hiện lũy thừa mô-đun phục vụ cho việc trao đổi khóa.
- Sinh các số nguyên tố lớn an toàn nhằm phục vụ các giao thức mật mã.
- Triển khai trao đổi khóa Diffie-Hellman[1], cho phép Alice và Bob tính toán một bí mật chung bằng cách sử dụng khóa riêng của họ.
- Viết báo cáo chi tiết về những phần đã tìm hiểu được trong bài tập này, cùng với các mô tả chi tiết về phần chương trình sinh viên viết.

Giới thiệu bài tập

Sinh viên được cung cấp một chương trình mẫu **chưa hoàn chỉnh** về cấu trúc chung của bài tập. Chương trình này chỉ nhằm mục đích mô tả các nhiệm vụ một các rõ ràng, **anh/chị có thể sử dụng mẫu này, hoặc tự viết một chương trình khác, miễn sao thực hiện được các nhiệm vụ được giao**. Tổng quan các nhiệm vụ:

1. Cài đặt hàm `modular_exponentiation` để tính lũy thừa mô-đun **một cách hiệu quả**.
2. Triển khai hàm `generate_safe_prime` để sinh các số nguyên tố lớn an toàn.
3. Triển khai hàm `generate_private_key` để sinh khóa riêng cho Alice và Bob.
4. Hoàn thành trao đổi khóa Diffie-Hellman.

[1] https://www.math.auckland.ac.nz/ sgal018/crypto-book/ch20.pdf

1 Chương trình mẫu

```
// A: Triển khai hàm lũy thừa mô-đun
// Hàm cần thực hiện: (base^exponent) % mod
<...> modular_exponentiation(<...> base, <...> exponent, <...> mod) {
    <...> result = 1;
    base = base % mod;

    // Ví dụ về logic lũy thừa mô-đun
    while (exponent > 0) {
        if (exponent % 2 == 1) {
            // Cập nhật kết quả với phép nhân mô-đun
        }
        // Cập nhật base và exponent cho lần lặp tiếp theo
    }
    return result;
}

// B: Triển khai hàm sinh số nguyên tố ngẫu nhiên
<...> generate_safe_prime(int bit_size) {
    // 1. Tự cài đặt logic để sinh một số nguyên tố an toàn
    // 2. Tự viết hàm kiểm tra nguyên tố (ví dụ: Miller-Rabin)

    <...> prime = 0;
    return prime;
}

// C: Triển khai hàm sinh khóa riêng ngẫu nhiên
<...> generate_private_key(<...> p) {
    // Sử dụng sinh số ngẫu nhiên để tạo khóa riêng
    // Khóa riêng nên nằm trong khoảng [2, p-2]
    <...> private_key = 0;
    return private_key;
}

// D: Hoàn thành logic trao đổi khóa Diffie-Hellman
int main() {

    // 1. Sinh số nguyên tố lớn p và phần tử sinh g
    int bit_size = 512; // Kích thước bit ví dụ, có thể điều chỉnh
    <...> p = generate_safe_prime(bit_size); // Sinh một số nguyên tố
    <...> g = 2; // Phần tử sinh, sinh viên cần tìm hiểu và chọn giá trị khác

    // 2. Sinh khóa riêng của Alice và Bob
    <...> a = generate_private_key(p); // Khóa riêng của Alice
    <...> b = generate_private_key(p); // Khóa riêng của Bob

    // 3. Tính giá trị công khai của Alice và Bob
    <...> A = modular_exponentiation(g, a, p); // Alice tính A = g^a % p
    <...> B = modular_exponentiation(g, b, p); // Bob tính B = g^b % p

    // 4. Tính bí mật chung
    <...> alice_shared_secret = modular_exponentiation(B, a, p); // Alice tính s = B^a % p
    <...> bob_shared_secret = modular_exponentiation(A, b, p); // Bob tính s = A^b % p

    // 5. Hiển thị kết quả và xác minh rằng bí mật chung trùng khớp
    std::cout << "Bí␣mật␣chung␣Alice␣nhận␣được:␣" << alice_shared_secret << "\n";
    std::cout << "Bí␣mật␣chung␣Bob␣nhận␣được:␣" << bob_shared_secret << "\n";
    std::cout << "Quá␣trình␣tính␣toán␣đúng␣không?␣" << (alice_shared_secret == bob_shared_secret) << "\n";

    return 0;
}
```

2 Yêu cầu cụ thể

2.1 Hàm lũy thừa mô-đun

Hoàn thành hàm `modular_exponentiation`:

$$\text{result} = (\text{base}^{\text{exponent}}) \mod mod$$

Sinh viên phải đảm bảo chương trình xử lý các số lớn một cách hiệu quả.

2.2 Sinh số nguyên tố

Triển khai hàm `generate_safe_prime` để sinh một số nguyên tố lớn[2] p, trong đó p là số nguyên tố và $\frac{p-1}{2}$ cũng là số nguyên tố.

Anh/chị cần tự cài đặt chương trình sinh số nguyên tố lớn, không sử dụng thư viện hỗ trợ. Các bạn có thể tham khảo các thuật toán kiểm tra số nguyên tố [3] như kiểm tra nguyên tố Miller-Rabin.

2.3 Sinh khóa riêng

Viết hàm `generate_private_key` để sinh ngẫu nhiên khóa riêng cho Alice và Bob. Khóa riêng phải là một số ngẫu nhiên trong khoảng $[2, p-2]$, trong đó p là số nguyên tố đã sinh trước đó. Độ lớn của p được xác định bằng số bit, có thể điều chỉnh trong code, tối thiểu 512 bit.

2.4 Trao dổi khóa Diffie-Hellman

Triển khai giao thức trao đổi khóa Diffie-Hellman, gợi ý cách làm trong hàm `main` của chương trình mẫu, hoặc các bạn có thể tham khảo các nguồn trên mạng[4]. Alice và Bob sẽ thực hiện:

- Tính giá trị công khai bằng hàm `modular_exponentiation`.
- Trao đổi giá trị công khai và tính bí mật chung bằng giá trị công khai nhận được.

Sau khi thực hiện các bước trên, sinh viên xác minh bí mật chung mà Alice và Bob tính được là trùng khớp.

2.5 Báo cáo

Báo cáo cần đảm bảo các phần sau:

1. **Hệ thống xử lý số lớn**: Bài tập yêu cầu thao tác trên số nguyên tố có độ lớn ít nhất 512 bit, vì vậy, nếu sinh viên có cài đặt một kiểu dữ liệu lớn riêng cho phần bài tập này, anh/chị cần mô tả chi tiết về kiểu dữ liệu đó, cách các toán tử của nó hoạt động.
2. **Hàm kiểm tra số nguyên tố**: Viết báo cáo những gì tìm hiểu được về các thuật toán kiểm tra số nguyên tố.
3. **Số ngẫu nhiên**[5]: Viết báo cáo những gì tìm hiểu được về tính ngẫu nhiên và độ quan trọng của nó trong mật mã học, cách tạo một hàm sinh số ngẫu nhiên an toàn trên máy tính.
4. **Trao đổi khóa Diffie-Hellman**: Tìm hiểu và trình bày một ứng dụng thực tế của giao thức này.
5. **Mô tả chương trình đã cài đặt**: Viết chi tiết cách chạy chương trình đã cài đặt, các thay đổi các tham số để điều chỉnh độ lớn của số nguyên tố hoặc các thành phần khác trong chương trình của sinh viên.
6. **Tự đánh giá**: Phân tích ưu/nhược điểm của chương trình đã cài đặt, cách cải thiện.

[2] https://planetmath.org/SafePrime
[3] https://crypto.stanford.edu/pbc/notes/numbertheory/millerrabin.html
[4] https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/
[5] Phần này không bắt buộc, được tính vào mục điểm cộng cho bài tập này

3 Các quy định khác

- Bài tập được theo **nhóm tối đa 4 sinh viên**.
- Thời gian thực hiện là 2 tuần tính từ lúc được đăng trên trang môn học.
- Sinh viên sử dụng ngôn ngữ C/C++
- Cấu trúc bài nộp:

```
<MSSV1-MSSV2-MSSV3-MSSV4>.zip
├── project_01_source
│   └── *.cpp
└── project_01_report
    └── *.pdf
```

  Trong đó:

  - `<MSSV1-MSSV2-MSSV3-MSSV4>` là các mã số sinh viên của nhóm sinh viên, sắp xếp theo thứ tự trong danh sách lớp.
  - `.zip` là định dạng nén cho bài làm (định dạng ZIP).
  - `project_01_source` là nơi chứa mã nguồn của bài tập.
  - `project_01_report` là nơi chứa (các) báo cáo ở định dạng pdf.
- Chương trình thực hiện yêu cầu nào không biên dịch được hoặc lỗi thì phần đó không có điểm.
- Thư viện chuẩn của `C++` (`C++` Standard Library) là các thư viện được liệt kê ở đây: `https://en.cppreference.com/w/cpp/standard_library`

  Cần lưu ý về phiên bản `C++` để lựa chọn thư viện thích hợp.
- Mọi thắc mắc vui lòng gửi về email: `ndhy@fit.hcmus.edu.vn`