Tên: Phan Trung Hiếu

MSSV: 19127404

Bài tập thực hành tuần 4 – Cryptograpy

Các hàm được sử dụng trong bài làm:

- **generateLargePrime(keysize)**: dùng để tạo ra một số nguyên tố lớn, với keysize là số bit của số đó(trong bài thực hành này chủ yếu tạo ra số nguyên tố khoảng 32 bit)
- **isPrime(n), rabinMiller(n,d,a):** 2 hàm này để kiểm tra một số có phải số nguyên tố hay không, thay vì dùng phương pháp thông thường thì thời gian check sẽ cao do số nguyên tố ta cần check có kích thước khá lớn, nên sẽ dùng thuật toán Rabin Miller để bổ sung tốc độ cho thuật toán

  Ý tưởng của thuật toán Miller:
  Xét số lẻ $n > 1$, đẻ kiểm tra n có phải là số nguyên tố hay không, ta làm như sau:
  ● Phân tích $n - 1$ thành dạng $2^s * d$ (s và d là số nguyên dương, d lẻ)
  ● Xét mỗi số nguyên tố a trong khoảng $[2, \min(n-1), [2(\ln n)^2]]$

  Nếu $a^d \neq 1\ (mod\ n) và\ \left(a^d\right)^{2^r} \neq -1\ (mod\ n) với\ mọi\ r$ từ 0 đến $s - 1$ thì n không phải là số nguyên tố

  ● Nếu n vượt qua tất cả các lần thử với a ở trên thì n là số nguyên tố

  Tuy nhiên thì trong thực tế, ta không cần phải xét tất cả các số a, mà chỉ cần dùng một vài số là đủ. Cụ thể, ta sẽ chỉ thử với a trong tập {2,3,5,7,11,13,17,19,23,29,31,37,41}

- **generateKeys()**: hàm này dùng để tạo ra cặp khoá bí mật và công khai p, q, e, d, N
- **gcd(a, b)**: tính ước chung lớn nhất theo thuật toán Euclid
- **bezout(a, b), modularInv(a, b)**: dùng để tính e theo thuật toán bezout
- **class RSA** gồm 2 hàm chính: ● **Encrypt(m)** theo công thức $c = m^e\ mod\ n$
  ● **Decrypt(c)** theo công thức $m = c^d\ mod\ n$

Để chạy chương trình, dùng các compiler python để chạy chương trình như bình thường

Tài liệu tham khảo:

- Slide bài giảng của giảng viên
- https://vietcodes.github.io/algo/miller (Thuật toán Miller)