Brno University of Technology
Faculty of Information Technology

Cryptography
2. Project – RSA

Dominik Harmim (xharmi00)
xharmi00@stud.fit.vutbr.cz
3rd May 2020

1 Introduction

The goal of this project is to implement the RSA algorithm — an asymmetry cryptography algorithm. The program that should be created should be able to generate parameters of the RSA, encrypt and decrypt messages, and break the algorithm using factorisation of the public modulus. The program is implemented in C++ and it is used arithmetic library GMP[1] for the computation with large numbers. In the following chapters, there are briefly described the methods and algorithms used. Appropriate bibliography and other sources are mentioned as well.

2 Encryption and Decryption

An encryption and decryption is implemented by [4]. The private transformation D and the public transformation E are defined as follows: $D(c) = m = c^d \mod n$; $E(m) = c = m^e \mod n$, where m is a decrypted message, c is an encrypted message, e is the public exponent, d is the private exponent, and n is the public modulus.

3 RSA Parameters Generation

The generation of RSA parameters is implemented according to [4]. At first, two random prime numbers (p and q) are generated and $n = p \cdot q$ and $h = (p - 1) \cdot (q - 1)$ is computed. Then, it is chosen e to be an integer in range $(2, h - 1)$ with $GCD(e, h) = 1$. Further, it is found the multiplicative inverse d of e, modulo h. Now, n and e are public and d, p, q, h are secret.

The computation of a greatest common divisor (the GCD function) is done using the Euclid's algorithm. The computation of a multiplicative inverse is done using the extended Euclid's algorithm. These algorithms are in detail described in [4].

[1] Arithmetic library **GMP** – https://gmplib.org.

For the generation of the prime numbers, it is used the probabilistic approach using Solovay-Strassen test with the computation of the Jacobi symbol. These algorithms are explained and demonstrated in [4, 5, 2].

4 Breaking the Algorithm

Breaking of the algorithm is achieved using the factorisation of the public modulus. Once the factorisation is done, one of the secret prime numbers is obtained and the other one is then easily calculated as well as the private exponent. So, the private key is obtained.

Factorisation of the public modulus is implemented using the Pollard Rho Brent Integer Factorisation, see [1, 3].

5 Conclusion

Within this project, the RSA algorithm has been successfully implemented in a desirable way. It was successfully tested on computer merlin.fit.vutbr.cz. It was experimentally verified that the program can correctly generate keys for the public modulus up to 4096 bits length and that it can break the algorithm for the public modulus up to 100 bits length quite fast. The program works properly also for longer keys but, as expected, it will take more time.

Bibliography

[1] Brent, R.: An improved Monte Carlo factorization algorithm. *BIT Numerical Mathematics*. vol. 20, no. 2. 1980: pp. 176 – 184. ISSN 0006-3835.

[2] Cook, J. D.: Computing Legendre and Jacobi symbols [online]. Feb 2019 [cit. 2020-05-03]. Retrieved from: https://www.johndcook.com/blog/2019/02/12/computing-jacobi-symbols

[3] fR0DDY: Pollard Rho Brent Integer Factorization [online]. Sep 2010 [cit. 2020-05-03]. Retrieved from: https://comeoncodeon.wordpress.com/2010/09/18/pollard-rho-brent-integer-factorization

[4] Nechvatal, J.: Public-Key Cryptography. Technical report. National Institute of Standards and Technology. Apr 1991. special publication.

[5] Wikipedia: Solovay–Strassen primality test [online]. Dec 2019 [cit. 2020-05-03]. Retrieved from: https://en.wikipedia.org/wiki/Solovay-Strassen_primality_test