C++ Implementation of Cryptography Algorithms

Ghaith Arar, Amanda Ly, *Student Member, IEEE,* and Neel Haria
Stevens Institute of Technology
Hoboken, NJ 07030, USA
Email: garar@stevens.edu, aly@stevens.edu and nharia@stevens.edu

***Abstract*—Cybersecurity has gotten increasingly significant in the past few years and with the rise in the widespread use of technology, there has also been a rise in cybercrime and cyber-attacks. These cybersecurity threats consist of but are not limited to malware, phishing, data leakage, hacking, structured query language injection, denial-of-service attacks, and domain name system tunneling. Cybersecurity is important because it pertains to protecting user's sensitive data and personal information; however, securing information is a challenge. Cryptography is an integral part of modern world information security to make the virtual world a safer place. Cryptography is a process of making information unintelligible to an unauthorized person. Hence, providing confidentiality to genuine users. This paper will look into various cryptography algorithms and implement the Rivest–Shamir–Adleman (RSA) algorithm, and Diffie-Hellman key exchange using C++.**

***Index Terms*—Rivest–Shamir–Adleman (RSA), Diffie-Hellman, Data Encryption, Cryptography, Data Decryption.**

I. Introduction

INFORMATION security plays an important role in protecting digital information against security threats and keeps information secret by protecting it from unauthorized access. Cryptography involves secret writing between two or more people so that others looking in cannot decipher what is being said or shared. There are two types of cryptosystems based on the number of keys involved: a symmetric key cryptosystem and an asymmetric key cryptosystem. Cybersecurity experts define cryptography in five components and the underlying technology of cryptography involves the encryption of plain text and the decryption of cipher text. Encryption is the process of conversion of data, called plain text, into an unreadable form, called cipher text, that cannot be easily understood by unauthorized people. Decryption is the process of converting encrypted data back into its original form, so that it can be understood by the people who are authorized to read the data [1]. The idea is that a recipient of plain text would not need special knowledge or equipment to understand what is being communicated, and the decryption algorithm decodes the cipher text to the original understandable plain text. This is all done with a set of keys - secret information known by members of the group that parameterize the encryption and decryption.

The remainder of this paper is organized as follows. Section II will discuss the related works pertaining to cryptography. Section III discusses the different cryptography algorithms. Section IV presents the team's novel implementation of RSA and Diffie-Hellman key exchange using C++. Section V will discuss the observations of our implementation and the paper will be concluded in section VI.

II. Related Work

Texttextext sample text Texttextext sample text Texttextext sample text Texttextext sample text Texttextext sample text

III. Cryptography Algorithms

TexttextextTexttextextTexttextextTexttextextTex

A. Symmetric Algorithms

blah blah knowledge part 1

1) AES: Advanced Encryption Standard

2) DES: Data Encryption Standard

B. Asymmetric Algorithms

Asymmetric encryption, also known as public-key cryptography, is a relatively new method, compared to symmetric encryption, and uses two keys to encrypt plain text. Asymmetric encryption was introduced to complement the inherent problem of the need to share the key in the symmetric encryption model, eliminating the need to share the key by using a pair of public-private keys. A public key is made freely available to anyone who might want to send you a message. The second private key is kept a secret so that you can only know. This algorithm uses a key generation protocol to generate a key pair so both the keys are mathematically connected with each other. The benefits of asymmetric cryptography is the elimination of key distribution, an increased security due to the lack of key transmission and the use of digital signatures. The main disadvantage is the slower computation/processing speed compared to symmetric cryptography. Types of asymmetric cryptosystem [2]. RSA, Elliptic Curve Cryptosystem (ECC), Diffie-Hellman, and Digital Signature Algorithm (DSA) are some of commonnly used asymmetric algorithms.

1) RSA: There were several schemes and algorithms proposed for public key cryptography since its founding, and one of the earlier known algorithm is RSA [3]. RSA, also known as the Rivest–Shamir–Adleman algorithm, was founded in 1977 and is the most common public key algorithm in cryptography world. RSA is named for its inventors, Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, who created it as faculty at the Massachusetts Institute of Technology. Although the concept of RSA is still widely used in electronic commerce protocol, RSA itself has many flaws in its design therefore not preferred for the commercial use [4]. The key size of RSA is greater than or equal to 1024 bits with a minimum block size of 512 bits. This asymmetric algorithm has a high power consumption, a slow encryption, a slow decryption and is the least secure when compares to the symmetric algorithms AES and DES mentioned above. It also has an inherent vulnerability to brute forced and oracle attacks.

Even though, applying the algorithm is relatively simple, it lies behind powerful math theorems to ensure its strength. The security of RSA relies on the difficulty of factoring the product of two large prime numbers. This algorithm is based on the theory of Prime Numbers and the fact that it is easy to find and multiply large prime numbers, but it is extremely difficult to factor their product. It's security depends on the difficulty of decomposition of large numbers. This paper will later show a basic implementation of RSA in Section IV using C++ and BigInteger Libraries GMP and Boost.

2) Diffie-Hellman: The Diffie-Hellman (DH) algorithm is one of the most important developments in public-key cryptography. It was founded between 1969-1973 but not published until 1976 by Whitfield Diffie and Martin Hellman researchers from Standford University and Ralph Merkle researcher from the University of California at Berkeley. The DH key exchange, a procedure which is one of the first public key cryptographic protocols, does not encrypt data, instead, it generates a secret key common to both the sender and the recipient. Although they never agreed on using a particular key, through mathematically linked processes the two parties can independently generate the same secret key and then use it to build a session key for use in asymmetric algorithm [5]. This method allows two parties that have no prior knowledge of each other to generate a shared private key with which they can exchange information across an insecure channel.

Similarly to RSA, Diffie-Hellman also uses prime numbers. The simplest and the original implementation [6] of the algorithm uses the multiplicative group of integers modulo p, where p is prime, and g is a primitive root modulo p. Next, both parties will pick a secret integer that they do not reveal to anyone else.

DH is usually utilized when you encrypt data on the Web utilizing either SSL (Secure Socket Layer) or TLS (Transport Layer Security) [7]. DH protocols can be attacked by denial-of-service attacks, outsider attacks, insider attacks and man-in-the-middle attacks. In a denial-of-service attack, the attack could delete the messages or overwhelm the parties with unnecessary computations or communication. In an outsider attack, the attacker tries to disrupt the DH protocol by adding, removing replaying messages or other similar methods so the attacker is able to obtain interesting information. An insider attack is one where one of the parties creates a breakable protocol run on purpose to gain knowledge about the secret key of the other party. In a man-in-the-middle attack, the attacker would intercept one parties public value and send their own value to the other party. The attacker would then substitute the public value of the second party with their own and send it back to the first party. This attacker would then be able to decrypt and modify the messages send by either party [8]. This vulnerability is present because Diffie-Hellman key exchange does not authenticate the participants.

3) ECC: Elliptic Curve Cryptosystem

4) DSA: Digital Signature Algorithm

IV. Implementation

this is where we put our code and explain it.

A. Pseudo Code of RSA:

int x = prime number
int y = prime number
int $\phi = x * y$
int e = find coprime(ϕ)
int d = $1\ mod\ (\phi)/e$
public_key = *(e,n)*
private_key = *(d,n)*

B. Steps of RSA

1) Generating the Keys:

i) Select two large prime numbers, x and y. The prime numbers need to be large so that they will be difficult for someone to figure out.
ii) Calculate $n = x * y$.
iii) Calculate the **totient** function $\phi(n) = (x-1)(y-1)$.
iv) Select an integer e, such that e is co-prime to $\phi(n)$ and $1 < e < \phi(n)$. The pair of numbers(n,e) makes up the public key.
v) Calculate d such that $e * d = 1\ mod\ \phi(n)$ or $d = e^{-1}\ mod\ \phi(n)$.

2) Encryption: Given a Plain-text P, represented as a number the Cipher text C is calculated as $C = P^n mod\ n$

3) Decryption: Using the private key(n,d) the plain-text can be found using $P = C^d mod\ n$

C. Steps of Diffie-Hellman

Let Alice be the first party and Bob be the second party.

1) Public Parameter Generation: A trusted party chooses and publishes a large prime number *p* and a nonzero integer *g* modulo *p*

2) Private Computations:

i) Alice chooses a secret integer a and then computes $A \equiv g^a \ mod \ p$.

ii) Bob chooses a secret integer b and then computes $B \equiv g^b \ mod \ p$.

3) Public Exchange of Values: Alice will send A to Bob and in exchange, Bob will send B to Alice.

4) Final Private Computations:

i) Since Alice now knows B, it will use its own secret integer a and compute $B^a \ mod \ p$.

ii) Likewise, Bob now knows A and with its own secret integer b will compute $A^b \ mod \ p$.

The shared secret value is $B^a mod p \equiv (g^b)^a \equiv g^{ab} \equiv (g^a)^b \equiv A^b \ mod \ p$.

D. GMP-Library

GMP is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating-point numbers. According to the library documentation, there is no practical limit to the precision except the ones implied by the available memory in the machine GMP runs on.

The main motivations behind GMP are cryptography applications and research. The library uses full words as the basic arithmetic type to achieve maximum speed. It uses highly optimized assembly code for the most common inner loops.

Although the library is known to work on Windows in both 32-bit and 64-bit mode, it's main target platform are Unix-type systems, such as GNU/Linux, Solaris, HP-UX, and Mac OS X/Darwin.

This project uses mainly the high-level signed integer arithmetic functions (mpz). There are about 150 arithmetic and logic functions in this category.

All declarations needed to use GMP are collected in the include file gmp.h. It is designed to work with both C and C++ compilers. Also, all programs using GMP must link against the libgmp library. On a typical Unix-like system this can be done with '*-lgmp*'. GMP C++ functions are in a separate libgmpxx library. This is built and installed if C++ support has been enabled.

In this project, among the multiple types GMP provides, we only used the integer type and GMP integer arithmetic functions. The C data type that GMP provides to represent multiple precision integers is *mpz_t*.

All GMP integer objects must be initialized before passing them to functions, and cleared when they are no longer needed. The GMP library provide multiple functions for Initialization and clearing. We mainly used **void mpz_init (mpz_t x)** which Initializes x, and set its value to 0. However, **void mpz_inits (mpz_t x,...)** can be used in the same fashion to initialize multiple objects at once. This function takes a NULL-terminated list of mpz_t variables

In a similar way, **void mpz_clear (mpz_t x)** and **void mpz_clears (mpz_t x,...)** can be used to free memory sapes occupied by GMP mpz_t objects.

The library offers multiple functions for assignment, however all such functions assume the GMP mpz_t objects have been initialized before calling the function. We list below the functions used in this project. For the full list we suggest visiting the GMP ;ibrary website.

void mpz_set (mpz_t rop, const mpz_t op)
void mpz_set_ui (mpz_t rop, unsigned long int op)
void mpz_set_si (mpz_t rop, signed long int op)

Initialization and assignment can be combined. GMP library provides functions a parallel series of initialize-and-set functions which initialize the output and then store the value there. It's noteworthy to mentioned that programmers should not use an initialize-and-set function on a variable already initialized.

void mpz_init_set (mpz_t rop, const mpz_t op)
void mpz_init_set_ui (mpz_t rop, unsigned long int op)
void mpz_init_set_si (mpz_t rop, signed long int op)

A crucial step of RSA algorithm and many other Cryptography algorithms is generating seemingly random and very large prime number. In general, for RSA, the greater the numbers used to generate the keys, the harder it takes to crack the algorithm. GMP library offers a good set of functions for arithmetic operations, division, exponentiation, random numbers, and primality test. The library offer two groups of function, older function that rely on a global state, and newer functions that accept a state parameter that is read and modified. This project uses the newer group only.

E. Design 1a: RSA with GMP Library

- insert design & explain it a little bit-

F. Design 1b: RSA with Boost

- insert design & explain it a little bit-

G. Design 2: Diffie-Hellman Key Exchange

- insert design & explain it a little bit-

V. Discussion

this is where we discuss our observations about our implementations and what future works can be done it.

VI. Conclusion

This paper is about ...

References

[1] A. Kahate, *Cryptography and network security*. Tata McGraw-Hill Education, 2013.

[2] K. Brush, L. Rosencrance, and M. Cobb, "What is asymmetric cryptography and how does it work?" 2020. [Online]. Available: https://searchsecurity.techtarget.com/definition/asymmetric-cryptography

[3] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[4] X. Zhou and X. Tang, "Research and implementation of rsa algorithm for encryption and decryption," in *Proceedings of 2011 6th international forum on strategic technology*, vol. 2. IEEE, 2011, pp. 1118–1121.
[5] D. A. Carts, "A review of the diffie-hellman algorithm and its use in secure internet protocols," *SANS institute*, pp. 1–7, 2001.
[6] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.
[7] S. Kallam, "Diffie-hellman: Key exchange and public key cryptosystems," *Master degree of Science, Math and Computer Science, Department of India State University, USA*, pp. 5–6, 2015.
[8] J.-F. Raymond and A. Stiglic, "Security issues in the diffie-hellman key agreement protocol," *IEEE Transactions on Information Theory*, vol. 22, pp. 1–17, 2000.