# C++ Implementation of Cryptographic Algorithms

By Amanda Ly, Ghaith Arar and Neel Haria

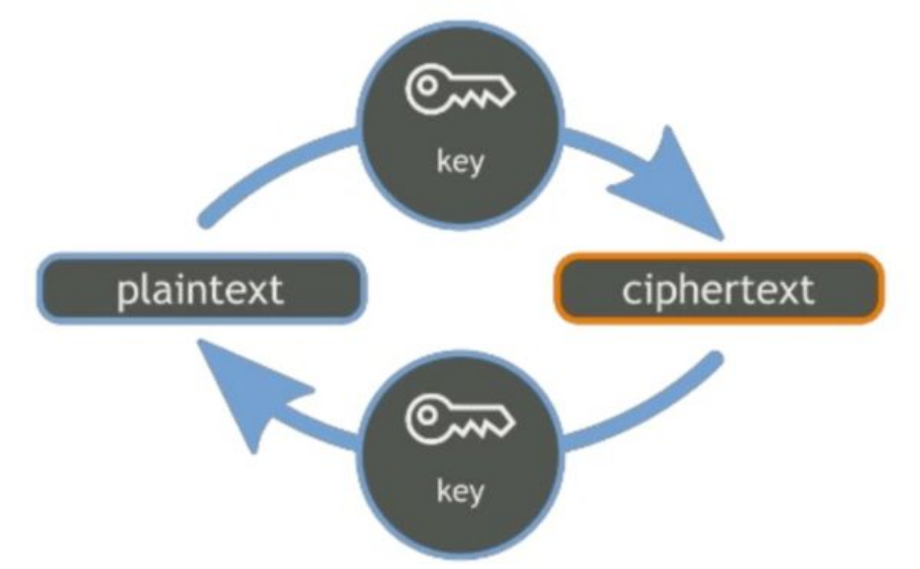
GitHub: CPE-593_FinalProject-RSA

# TABLE OF CONTENTS

# Overview

- What is cryptography?
- Why is it important?

“**Cryptography** is the study of secure communications techniques that allow only the sender and intended recipient of a message to view its contents.”



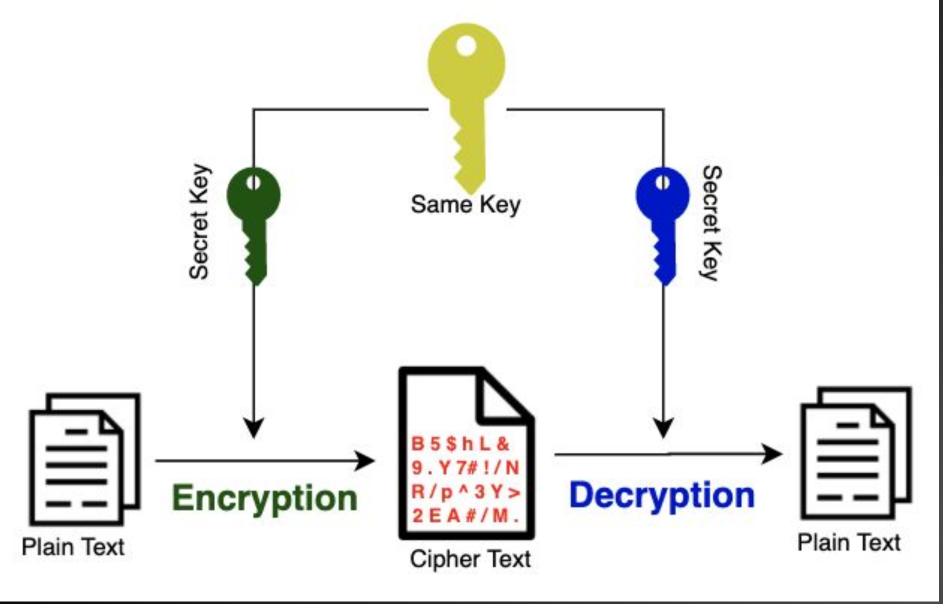

Importance:

- Confidentiality
- Identification and Authentication
- Integrity
- Non-repudiation

# 02

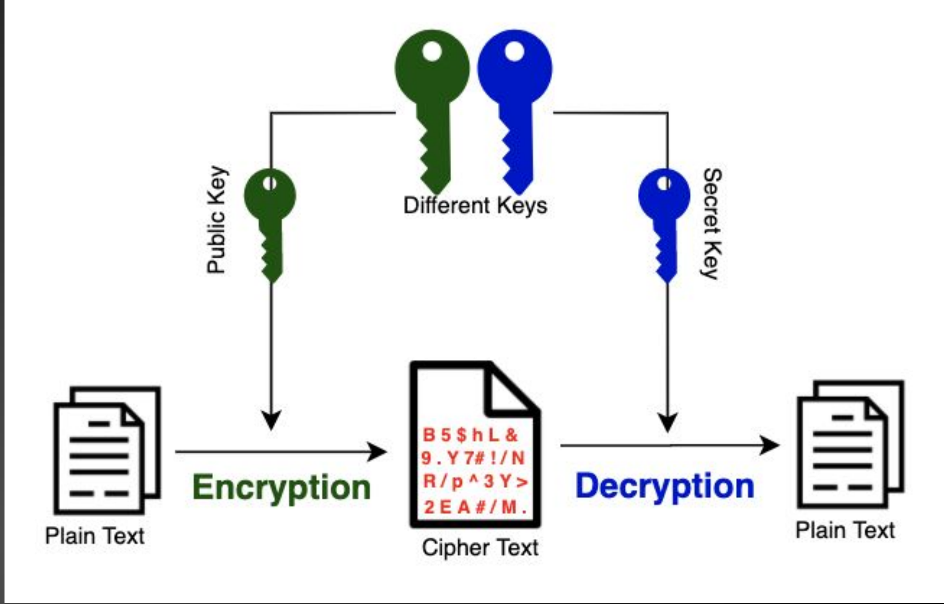
# Cryptographic Algorithms

- Asymmetric vs Symmetric




## Symmetric Encryption
## (Private Key)

- Advanced Encryption Standard (AES)
- Data Encryption Standard (DES)

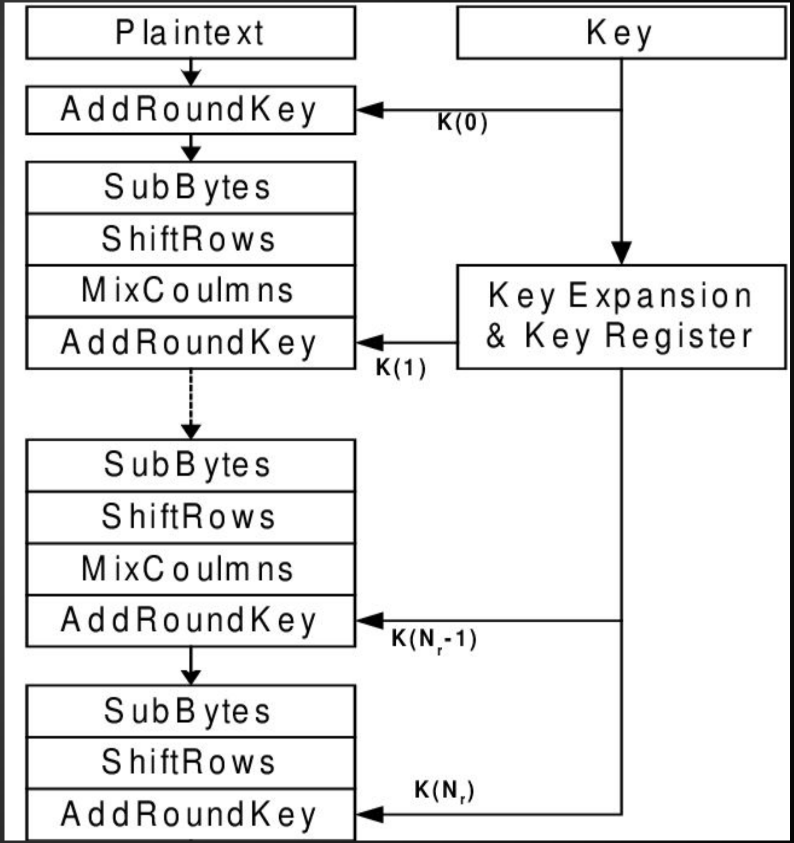



## Asymmetric Encryption
## (Public Key)

- Rivest–Shamir–Adleman (RSA)
- Diffie-Hellman (DH)
- Elliptic Curve Cryptosystem (ECC)
- Digital Signature Algorithm (DSA)

# Advanced Encryption Standard (AES)

- Developed by Vincent Rijmen and Joann Daeman
- Recommended by NIST to replace DES in 2001
- Algorithm is referred to as AES-128, AES-192,or AES-256, depending on the key length
- Relatively fast encryption and decryption
- Low power consumption
- Excellently secured



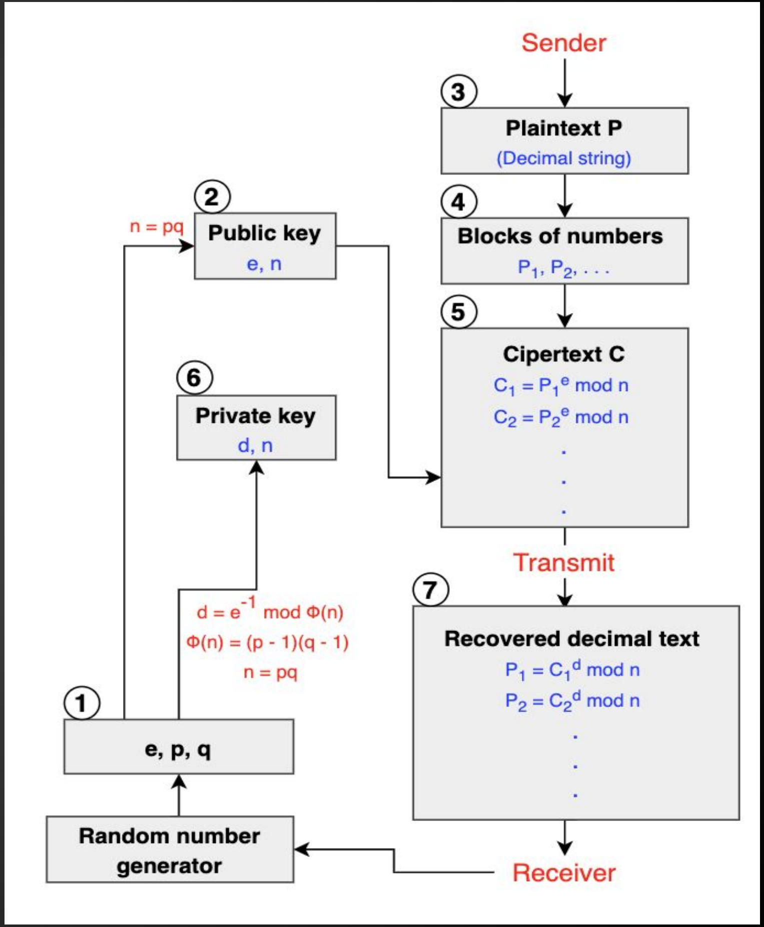
# 03

# Implementation

- RSA & Diffie-Hellman Key Exchange
- The math behind them

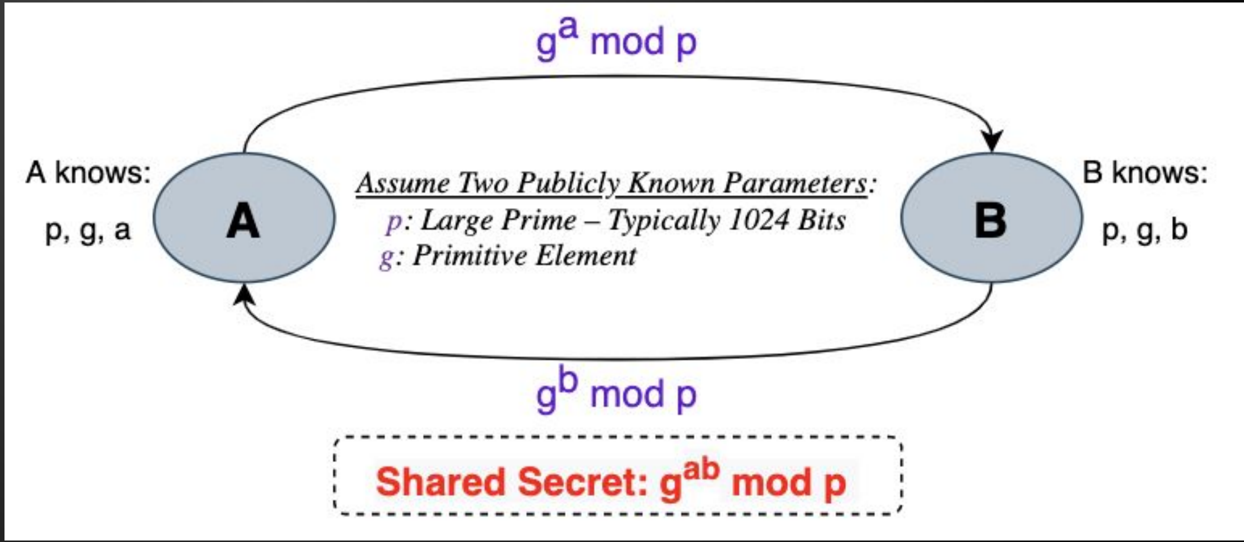
# Rivest–Shamir–Adleman (RSA)

- Founded in 1977 by Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman
- High power consumption
- Slow encryption and decryption
- Inherent vulnerability to brute force and oracle attacks
- Algorithm is based on the theory of Prime Numbers



# Diffie-Hellman Key Exchange (DH)

- Founded between 1969-1973; published in 1976 by Whitfield Diffie and Martin Hellman
- Some common DH protocols attacks are
  - denial-of-service attacks
  - man-in-the-middle attacks
  - outsider attacks
  - insider attacks



# 04

# Designs

- Language: C++
- Using BigInteger Library GMP

# Design: Diffie-Hellman using GMP

```
// Below:
// Alice choose a = kpr,A ∈ {2, . . . , p−2}
// Alice compute A = kpub,A ≡αa mod p
// a is Alice's secret key
// Alice computes A , and sent it to Bob
    mpz_t a;
    mpz_t A;
    mpz_inits(a, A, NULL);

    gmp_randstate_t state_a;
    gmp_randinit_mt (state_a);
    gmp_randseed_ui(state_a, time(NULL) );
    mpz_urandomm(a, state_a, n2);       //choose a = kpr,A ∈ {2, . . . , p−2}

    mpz_powm(A, alpha, a, p);       //compute A = kpub,A ≡α^a mod p

    cout << "Alice secret key a: " << endl;
    gmp_printf(" %Zd \n", a);
    cout << endl;

    cout << "Alice public key A: " << endl;
    gmp_printf(" %Zd \n", A);
    cout << endl;
```

```
// Bob choose b = kpr,B ∈ {2, . . . , p−2}
// Bob compute B = kpub,B ≡αb mod p
// b is Bob's secret key
// Bob computes B , and sent it to Alice
    mpz_t b;
    mpz_t B;
    mpz_inits(b, B, NULL);

    gmp_randstate_t state_b;
    gmp_randinit_mt (state_b);
    gmp_randseed_ui(state_b, (time(NULL) + 50) );
    mpz_urandomm(b, state_b, n2);       //choose b = kpr,B ∈ {2, . . . , p−2}

    mpz_powm(B, alpha, b, p);       //compute B = kpub,B ≡αb mod p


    cout << "Bob secret key b: " << endl;
    gmp_printf(" %Zd \n", b);
    cout << endl;

    cout << "Bob public key B: " << endl;
    gmp_printf(" %Zd \n", B);
    cout << endl;
```

* These are only code snippets. Full code on GitHub*

# Design: RSA using GMP

* These are only code snippets. Full code on GitHub*

We also wrote RSA using Boost Library but GMP is more efficient.

```
void generate_keys(mpz_t p, mpz_t q) {
    if (mpz_probab_prime_p(p, 50) == 2 && mpz_probab_prime_p(q, 50) == 2){
        if (p == q){
            cout << " Cannot have identical prime numbers"<<endl;
            return;
        }

        mpz_t phi, check;
        mpz_inits(phi, e, check, d, NULL);
        mpz_sub_ui(p, p, 1);
        mpz_sub_ui(q, q, 1);
        mpz_mul(phi, p, q);

        gmp_randstate_t rd;
        gmp_randinit_mt(rd);
        unsigned long int seed = 20201204;
        gmp_randseed_ui (rd, seed);

        mpz_urandomm(e, rd, phi);
        while (mpz_cmp(e, phi) < 0){
            mpz_gcd(check, e, phi);

            if (mpz_cmp_ui(check, 1) == 0){
                break;
            }
            else {
                seed = seed + 3;
                gmp_randseed_ui (rd, seed);
                mpz_urandomm(e, rd, phi);
            }
        }
        mpz_invert(d, e, phi);
        mpz_clears(check, phi, NULL);
    }
}
```

```
void encryptMsg(mpz_t pbk, string msg, mpz_t n){
//    For encryption, c = m^e mod n, where m = original message.
    mpz_t ciphertxt, x;
    unsigned long int tst;
    mpz_init(ciphertxt);

    for (int i = 0; i < msg.length(); i++)
    {
        mpz_init_set_ui(x, int(msg.at(i)));
        mpz_powm(ciphertxt, x, e, n);
        tst = mpz_get_ui(ciphertxt);
        arr.push_back(int(tst));

        cout << tst;

    }
    cout << endl;
    mpz_clears(x, ciphertxt, NULL);
    cout << endl;
}
```

```
void decryptMsg(mpz_t prvk, mpz_t n){
//    For decryption, m = c^d mod n.
    mpz_t OG, c, temp;
    mpz_t N1, N2;
    mpz_inits(N1, N2, NULL);
    mpz_init_set(N1, prvk);
    mpz_init_set(N2, n);

    mpz_init(c);
    long int temp2 = 0;

    for (int i = 0; i < arr.size(); i++){
        temp2 = arr[i];
        mpz_init_set_ui(temp, temp2);
        mpz_set_ui(c, temp2);
        mpz_powm(OG, temp, N1, N2);

    unsigned long int tst2;
    tst2 = mpz_get_ui(OG);
    cout << static_cast<char>(tst2);
    }
    cout <<endl;
}
```

# Output: RSA using GMP

```
ghaith@ubuntu:~/Documents/CPE_593/Final_Project$ g++ main.cpp -lgmpxx -lgmp
ghaith@ubuntu:~/Documents/CPE_593/Final_Project$ ./a.out
211 229 48319
*** Generating public/private key ***
Public Key: 19307
Private Key: 7043

Original Message:
THIS IS A TEST
... Encrypting message ...
Encrypted Message: 465394193627104381427692710438142769115872769465398696438144653 9


... Decrypting message ...
Decrypted Message:
THIS IS A TEST
Private Key: 7043
ghaith@ubuntu:~/Documents/CPE_593/Final_Project$ ./a.out
211 59 12449
*** Generating public/private key ***
Public Key: 2923
Private Key: 4867

Original Message:
THIS IS A TEST
... Encrypting message ...
Encrypted Message: 5110604917322442626173224426261092626511012108224451 10


... Decrypting message ...
Decrypted Message:
THIS IS A TEST
Private Key: 4867
ghaith@ubuntu:~/Documents/CPE_593/Final_Project$ ./a.out
```

# 05

# Discussion

- Weaknesses in the algorithms
- Weaknesses in the GMP Library

**Mathematical Attacks against RSA**

- By determining the prime factors p and q of the modulus n, an attacker can find out $\phi(n) = (p-1)(q-1)$, which is in turn enables to determine $d = e$ ^−1 (mod $\phi(n)$).

- By figuring out the totient $\phi(n)$ directly without first determining p and q from which $d = e$ ^−1 (mod $\phi(n)$ ) can be determined.

- By calculating d directly without first determining $\phi(n)$.

**Padding**

- RSA encryption is deterministic, i.e., for a specific key, a particular plaintext is always mapped to a particular ciphertext. An attacker can derive statistical properties of the plaintext from the ciphertext

- A possible solution to all these problems is the use of padding, which embeds a random structure into the plaintext before encryption and avoids the above mentioned problems. There is many techniques for padding.

## Side-Channel Attacks (Timing Attack)

- They exploit information about the private key which is leaked through physical channels such as the power consumption or the timing behavior.

- There are several countermeasures available to prevent the attack. A simple one is to execute a multiplication with dummy variables after a squaring that corresponds to an exponent bit 0. This results in a power profile (and a run time) which is independent of the exponent.

- Countermeasures against more advanced side-channel attacks are not as straightforward.

- GMP Library provides some low-level functions for cryptography where resilience to side-channel attacks is desired.

- In general, the security of a cryptosystem is a function of two things: the strength of the **algorithm** and the **length of the key**.
- A **perfect cryptography** algorithm is the one that no way to break it than with a brute-force attack.
- RSA with short secret key is proven **insecure** against brute force attack.
- This attack can be easily circumvented by choosing **large key**.
- However, the larger key will make the encryption and decryption process little **slow** as it will require **greater computations** in key generation as well as in encryption/decryption algorithm.
- As of 2020, the largest publicly known factored RSA number was 829 bits (250 decimal digits, RSA-250).[29] Its factorization, by a state-of-the-art distributed implementation, took approximately 2700 CPU years.
- **Quantum Computing** is the current biggest threat to RSA and many other cryptography algorithms.

**Recommendations:**

- In practice, RSA keys are typically 1024 to 4096 bits long.
- As of 2020, it is not known whether such 1024-bit keys can be cracked, but minimum recommendations have moved to at least 2048 bits.

# Conclusion

# Future Works

- RSA is not robust as it should be due to the deterministic property, it would be interesting to read and work on different methods to make it more secure.
- Implement Encryption methods on different file types.
- Hybrid encryption method utilizing multiple algorithms to improve the security.

# THANKS!

* For more information, review the notes section of each slide or our paper*