TECH7009 Research Proposal on RSA encryption and other Cryptographic schemes

Mohammad Ali Khan 19129163

Contents

1 Abstract

Investigation into research on RSA Encryption and its underlying mathematics and associated security issues, with some comparison to ECC and ElGamal encryption schemes

2 Introduction

This dissertation will attempt to perform an analysis on existing and current research material pertaining to RSA encryption, and through this analysis, identify areas which are well covered by current research and areas which are not as well covered, and propose areas in which further research could be done. Other encryption schemes will also be touched upon for the sake of comparison and to see if there are any interesting insights or research areas in there.

This research is important due to the wide use of RSA encryption- this encryption algorithm sees use in OpenSSL, was used in PGP encryption

It also has a number of features which makes secure and proper implementation difficult. These shall be covered in detail in the section "Real-world implementation of RSA"

3 RSA Cryptosystem

The RSA (Rivest-Shamir-Adelman) cryptosystem is an **asymmetric** cryptographic system, where anyone can use a shared **public key** in order to encrypt data, but only those who hold the appropriate corresponding **private key** can decrypt the encrypted data.

It is named after computer scientists Ronald Rivest, Adi Shamir, and Leonard Adelman, who published a paper on the system in 1977. However, a very similar system based on similar principles had been developed in 1973 by Clifford Cocks, a mathematician and cryptographer employed by the United Kingdom Government Communications Headquarters and was classified information until 1997 (Cocks 1973).

This cryptosystem exploits properties of group theory, modular arithmetic (and specifically **modular exponentiation**), semiprimes (natural numbers which are the products of two prime numbers), and Fermat's Little Theorem in order to function.

3.1 RSA encryption and decryption (as described)

This section will summarize the steps of sending and receiving a message in the RSA cryptosystem as per how it is described in the original paper by Rivest, Shamir and Adelman (R.L Rivest 1978). The differences between the described procedure of RSA and how it is actually implemented in practice will be discussed in a later section. A worked example will also be provided which will also explain the **extended Euclidean algorithm** and the process of **modular exponentiation**):

3.1.1 Key generation

In this first stage, the encryption and decryption keys are generated.

1. Two large distinct prime numbers, p and q are randomly selected.
2. The product N of these two large prime numbers, $N = p \cdot q$ is calculated. This product N is part of the *public key*.
3. The **Euler totient function** for N, or $\varphi(N)$ is calculated. For $N = p \cdot q$ where p and q are primes, this is defined as $\varphi(N) = (p-1) \cdot (q-1)$. $\varphi(N)$ is kept *private*.
4. An integer e is selected such that $2 < e < \varphi(N)$ and $gcd(e, \varphi(N)) = 1$, in other words- e and $varphi(N)$ are *co-prime* (have no common divisors apart from 1). This number is our *encryption key* and is part of the *public key*.
5. An integer d is calculated such that $e \cdot d \equiv 1(mod\varphi(N))$. This means that d is the *multiplicative inverse* of e in modulus $\varphi(N)$. This can be calculated using the *extended Euclidean algorithm*. This is the *decryption key* and is also kept *private*.
6. The public key (N, e) is transmitted out via a reliable and not necessarily secret route. The private key (d) is never transmitted, and kept with the receiver.

3.1.2 Encryption

Once a sender receives the public key, they can use it to send a message M.

1. The message M is expressed as an integer m such that $0 \leq m < N$.
2. The ciphertext c is computed using m and the encryption key e such that $c \equiv m^e(modN)$, then sent. Modular exponentiation is used at this step, which will be explained later.

3.1.3 Decryption

Once the receiver receives the ciphertext c, they can decrypt it:

1. retrieve m by calculating $c^d \equiv (m^e)^d \equiv m(modN)$. Now it can be read.

A worked example will be shown on the next page

3.1.4 A worked example of RSA encryption

Here is a demonstration of RSA encryption with very small values for easy demonstration of some of the calculations and algorithms involved

1. For a rudimentary demonstration, take two small primes $p = 13$ and $q = 31$.
2. For these values of p and q, $N = p \times q = 13 \times 31 = 403$.
3. The Euler totient is $\varphi(N) = (p-1) \times (q-1) = 12 \times 30 = 360$.
4. Pick a value e that is co-prime with $\varphi(N)$. We can pick any of the following values:

   7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 49, 53, 59, 61, 67, 71, 73, 77, 79, 83, 89, 91, 97, 101, 103, 107, 109, 113, 119, 121, 127, 131, 133, 137, 139, 143, 149, 151, 157, 161, 163, 167, 169, 173, 179, 181, 187, 191, 193, 197, 199, 203, 209, 211, 217, 221, 223, 227, 229, 233, 239, 241, 247, 251, 253, 257, 259, 263, 269, 271, 277, 281, 283, 287, 289, 293, 299, 301, 307, 311, 313, 317, 319, 323, 329, 331, 337, 341, 343, 347, 349, 353, 359

5. For the sake of this example, pick $e = 7$. This will be the encryption key.

Find a value d such that $d \cdot e \equiv 1 mod \varphi(N)$. This value is guaranteed to exist when $gcd(e, \varphi(N)) = 1$ and can be found using the **extended Euclidean algorithm** (the following description of the extended Euclidean algorithm comes from https://www.extendedeuclideanalgorithm.com/index.php ("The Extended Euclidean Algorithm" n.d.))

1. Start with the values $a = 360, b = 7$.
2. calculate ab which gives you a quotient $q = 51$ and a remainder $r = 3$.
3. Create a table with columns $a, b, q, r, s_1, s_2, s_3, t_1, t_2, t_3$
4. For the first row of this table, set a, b, q, r to the values calculated in steps 1 and 2, and set $s_1 = 1, s_2 = 0, s_3 = 1, t_1 = 0, t_2 = 1$
5. calculate $t_3 = t_1 - qt_2 = 0 - (51 \times 1) = -51$
6. for the second row, set a, b equal to the first row's values of b, r
7. calculate new values for q, r based on these new values of a, b. For $a = 7, b = 3$, we get $q = 2, r = 1$.
8. The second row's values of s_1, s_2 are equal to the first row's values of s_2, s_3.
9. The second row's values of t_1, t_2 are equal to the first row's values of t_2, t_3

a	b	q	r	s_1	s_2	s_3	t_1	t_2	t_3
360	7	51	3	1	0	1	0	1	-51
7	3	2	1	0	1	-2	1	-51	103
3	1	3	0	1	-2	7	-51	-103	258

Tab. 1: the Extended Euclidean algorithm

10. for the second row, $s_3 = s_1 - q \cdot s_2 = 0 - 2(1) = -2$ and $t_3 = t_1 - q \cdot t_2 = 1 - 2(-51) = 103$.
11. Steps 6 through 10 are repeated for the third row, which has $r = 0$.
12. this means that the desired value, the **multiplicative inverse** of 7, is equal to the second row's value for $t_3 = 103$
13. To confirm that 103 is the multiplicative inverse of 7 in modulus 360, calculate $7 \times 103 = 721$. Since $360 \times 2 = 720$, it can be concluded that $721 \equiv 1(mod 360)$ since $7(103) - 2(360) = 1$.

Having chosen a value for e and calculating the corresponding value for d, encryption and decryption is possible- Take an example message M comprising of a single character. This character has a value of 43. Encrypt it by calculating $C = M^e(modN)$. This can be done using **modular exponentiation**

The way to perform modular exponentiation of large numbers (as RSA would require) in a memory efficient manner is to take advantage of the identity $(a \cdot b)modm \equiv ((amodm) \cdot (bmodm))modm$, a simple version of it is as follows:

to calculate the value of $a^b modm$

1. set $c = 1, b' = 0$
2. set $c = a \cdot cmodm$
3. if $b' < b$ go to step 2, else $c \equiv a^b modm$

Let's apply it to our values of M, e:

1. $b_0 = 1; c_0 = 1 \cdot 43 \bmod 403 \equiv 43$
2. $b_1 = 2; c_1 = 43 \cdot 43 \bmod 403 \equiv 237$
3. $b_2 = 3; c_2 = 237 \cdot 43 \bmod 403 \equiv 116$
4. $b_3 = 4; c_3 = 116 \cdot 43 \bmod 403 \equiv 152$
5. $b_4 = 5; c_4 = 152 \cdot 43 \bmod 403 \equiv 88$
6. $b_5 = 6; c_5 = 88 \cdot 43 \bmod 403 \equiv 157$
7. $b_6 = 7; c_6 = 157 \cdot 43 \bmod 403 \equiv 303$

The original plaintext message of $M = 43$ has now been encoded as the ciphertext $C = 303$. It can now be transmitted.

There's a way to reduce the number of operations involved by exploiting a property of exponents: take for example the calculation of 3^9. This can be expressed as $3^2 \cdot 3^2 \cdot 3^2 \cdot 3^2 \cdot 3$- instead of multiplying 3 by itself 9 times, you square it 4 times and then multiply the result by 3 (5 operations as opposed to 9). This version of the modular exponentiation algorithm can be implemented using the following C/C++ code:

```
int modular_exp (int base, int exp, int mod) {
    // b^0 == 1;
    if (exp == 0) { return 1; }
    // b^1 == b
    int result = 1;
    if (exp == 1) { result = base; }
    while(1) {
        // if exp == 0, stop
        if (!exp) {break;}
        // integer halving of exp
        exp >>= 1;
        base = (base * base) % mod;  //squaring
        if (exp & 1) { result = (result * base) % mod; }
    }
    return result;
}
```

To decrypt this enciphered message 303, the receiver must perform modular exponentiation of the encoded number using the decryption key 103.

To show the calculation of $303^{103} \bmod 403$, the optimized version of the modular exponentiation algorithm is shown:

1. set $b_0 = 103, a_0 = 303, c_0 = 303$
2. $b_1 = floor(103/2) = 51$
3. $a_1 = 303^2 \equiv 328 \bmod 403$
4. $c_1 = 303 \cdot 328 \equiv 246 \bmod 403$
5. $b_1 = 51, a_1 = 328, c_1 = 246$
6. $b_2 = floor(51/2) = 25$
7. $a_2 = 328^2 \equiv 386 \bmod 403$
8. $c_2 = 246 \cdot 386 \equiv 251 \bmod 403$

9. $b_2 = 25, a_2 = 386, c_2 = 251$
10. $b_3 = floor(25/2) = 12$
11. $a_3 = 386^2 \equiv 289 \bmod 403$
12. since b_3 is even, $c_3 = c_2$
13. $b_3 = 12, a_3 = 289, c_3 = 251$
14. $b_4 = floor(12/2) = 6$
15. $a_4 = 289^2 \equiv 430 \bmod 403$
16. since b_4 is even, $c_4 = c_3$
17. $b_4 = 6, a_4 = 100, c_4 = 251$
18. $b_5 = floor(6/2) = 3$
19. $a_5 = 100^2 \equiv 328 \bmod 403$
20. $c_5 = 251 \cdot 328 \equiv 116 \bmod 403$
21. $b_5 = 3, a_5 = 328, c_5 = 116$
22. $b_6 = floor(3/2) = 1$
23. $a_6 = 328^2 \equiv 386 \bmod 403$
24. $c_6 = 251 \cdot 386 \equiv 43 \bmod 403$
25. $b_6 = 1, a_6 = 386, c_6 = 43$
26. finally, we get $c = 43$

(**NOTICE:** There may be errors in this demonstration).

As shown, performing modular exponentiation on the contents of the enciphered message C using the decryption key d results in the original plaintext M, or $C^d \equiv MmodN$

3.2 Explanation

The encryption and decryption operation in RSA works because of **Fermat's Little Theorem**. Fermat's little theorem states that for a prime number p and any integer a, $a^p - a$ is an integer multiple of p. This can be expressed in modular arithmetic as $a^p \equiv a(modp)$.

The security of RSA lies in the difficulty of integer factorization of very large numbers with current hardware and computing power at the time of this writing- In 2019, Fabrice Boudot, Nadia Heninger et al. factored a 240-digit number (RSA-240) using approximately 900 core-years of (standard, non-quantum) computing power, and estimated that factoring a 1024-bit RSA number would take about 500 times as long (F. Boudot 2020).

3.3 Real-world implementation of RSA

Real world implementations of RSA encryption differ from the way it is described in the original paper by Rivest et al.

This section will briefly describe some key differences between the description and implementation of RSA encryption, as well as talk about security risks and challenges that occur when implementing RSA:

3.3.1 Key Length/Size

In the context of RSA, *"Key length"* (or Key size) refers to the size in bits of pq, the modulus for the encryption and decryption keys.

In 2003, Schneier wrote in Applied Cryptography that a 2048-bit RSA key would be sufficient to keep data confidential for around 20 years, and was already suggesting 4096-bit keys (Schneier 2003).

Representing such large numbers in a program is difficult, and cannot be done with the default numeric types available in programming languages such as C and C++. The largest integer data type in C and C++ for example is

```
unsigned long long int
```

Which is 64 bit and would give you a maximum value of $18,446,744,073,709,551,615$ ("Fundamental types" n.d.).

instead custom data types/classes are written for representing these numbers (One such trick effectively involves representing numbers in base 2^32, by having them be represented by an array of integers with each "digit" being a number from 0 to $(2^32 - 1)$.

Implementation can get even more difficult with smaller scale devices such as those being used for Internet of Things (IoT) where processing power and resources are scarce. Many of them are restricted to 1024-bit RSA and require other tricks to keep them secure.

3.3.2 Key Selection

In actual RSA implementations the encryption key e is selected first, usually from a list of known primes with favored properties (Kaliski 1998). Then afterwards two distinct primes p and q are selected such that e is coprime with both $(p-1)$ and $(q-1)$.

In practice, the Fermat Primes (prime numbers of the form $2^(2^n)+1$) 3, 17 and 65537 (known as F_0, F_2 and F_4) are usually chosen, as these values make computing modular exponentiation faster, especially $e=3$, which is vulnerable to attack by use of Hastad's Attack (Chennagiri 2017).

3.3.3 Prime number selection

Prime number selection is a big source of the difficulties in proper implementation of RSA.

When selecting values for p and q probabilistic algorithms are usually used- these focus on generating a random odd number and using a Primality Test (also probabilistic) in order to check if it is prime (Xin Zhou 2011). More discussion on the role of randomness will be found in a later section, this section will focus on the selection of prime numbers.

A number of security weaknesses emerge when prime numbers are improperly selected for RSA encryption- the recommendation for proper implementation is that the prime numbers must be *globally unique* across an entire system, if $p$or q is ever reused in another modulus, both can be recovered by using a GCD algorithm ("Seriously, stop using RSA" 2019).

A study by Heninger et al in 2012 showed that nearly 1% of TLS traffic is vulnerable to this sort of GCD based attack owing to reuse of prime numbers (Nadia Heninger 2012).

3.3.4 Padding

RSA encryption in it's "textbook" form is susceptible to a number of different attacks, such as the "CCA2 attack" demonstrated by Knockel et al in "When Textbook RSA is Used to Protect the Privacy of Hundreds of Millions of Users" which involves discovering the AES key of an Android browser one bit at a time using transformed ciphertexts(J. Knockel 2018).

In most other real world cases, some form of "padding" scheme is applied to RSA encryption- this involves the use of hashing algorithms and a Feistel network,and the addition of random bits to make sure a message is a particular size, one such scheme is called OAEP and was proposed by cryptographers Bellare and Rogaway in 1995 (M. Bellare 1995).

3.3.5 Random Number Generation and Entropy

Random Number Generation (RNG) is an essential part of many cryptosystems for the generation of keys, salts, nonces, one-time pads etc.

True randomness would be acquired from something like radioactive decay or cosmic background radiation or even the state of bubbles in a Lava lamp (as performed by Silicon Graphics' Lavarand random number generator (n.d.)). When the hardware to acquire "true" randomness like this is unavailable, most computers use Pseudo-random number generators (PRNGs) based on performing repeated calculations on an internal seed using (ideally) unpredictable inputs such as mouse clicks or key presses or even other RNGs.

A Pseudo-random number generator must be "Cryptographically secure" if it is to be used for Cryptography. For a PRNG to be "cryptographically secure" it must:

1. pass **the Next-bit test:** if an attacker knows the first **k** bits of a PRNG, they should not be able to predict the **k + 1**st bit. This is a specific case of Yao's Test- an attacker should not be able to tell that a sequence was generated deliberately or procedurally (Yao 1982).
2. withstand **state compromise extensions**- an attacker should be unable to reconstruct previous random numbers from just successfully guessing the internal state of a PRNG

3.3.6 Computing and processing power

Computing and processing power of devices plays a big role in the way RSA is actually implemented in the real world, particularly in situations involving things like smartcards and other low power computing situations.

This causes real-world implementations of RSA to employ a number of optimizations and shortcuts which sometimes result in security issues- such as the case of the ROCA vulnerability which arose when key generation was restricted to primes of a particular form in order to speed up the process of key generation ("Seriously, stop using RSA" 2019).

This is also the reason a number of applications (such as cryptocurrency) chose to eschew RSA in favor of less power-hungry mechanisms such as Elliptic Curves (covered in more detail in a later section).

4 Attacks on RSA

4.1 Hastad Attack

a Hastad Broadcast attack is a type of attack that can be carried out on communications performed using the RSA cryptosystem. It relies on intercepting multiple messages sent using the same (small) encryption exponent (such as 3).

Suppose a sender sends the same message M to a group of people $P_1; P_2; ...P_k$ using the same encryption exponent ($e = 3$) and different moduli (N_i, e). Suppose then that an attacker intercepts the ciphertexts C_1, C_2, C_3 where $C_i \equiv M^3 mod N_i ...$ They can use the *Chinese Remainder Theorem* to compute a value C such that $C_i \equiv C mod N_i$, and then compute the cube root of C to obtain M (Chennagiri 2017).

4.2 Side Channel attacks

Side channel attacks are a class of attack that rely on information gathered from the way a computer protocol is implemented (hardware)- they rely on monitoring *physical* effects, such as acoustics, electromagnetic pulses, and electrical power levels.

Monitoring and performing signal analysis on electromagnetic pulses or signals from a system in order to deduce what the communications are is also known as Van Eck Phreaking.

This type of attack is of particular importance to implementations of RSA running on smartphones or devices such as card readers or other financial devices. A report from 2013 shows that it was possible to attack RSA encryption running on an Intel Atom processor this way (Anh Do 2013).

It must be noted however that certain variants of these attacks require laboratory grade equipment and close proximity to a target device, limiting their usefulness to a potential attacker, and that the security issues that were exploited in the previously cited paper have since been patched.

4.3 Quantum Computing/Shor's Algorithm

There is a growing amount of literature about the use of Quantum Computing and Shor's algorithm to factor large integers. Much of this literature focuses on how to optimize the quantum circuit used for the Quantum step of Shor's algorithm.

For example, in 2021, Dattani and Bryans used a multiplication table and other optimizations and number properties to factorize 56153 using quantum computing (Nikesh S. Dattani 2021). Without these kinds of optimizations and exploitation of number properties, the largest numbers factored by a quantum computer in the literature seems to be 91 (R. Selvarajan n.d.).

Quantum computing relies on mechanics differing from standard computing and thus is rife with challenges. Researchers such as Chennagiri (Chennagiri 2017) and others opine that Quantum Computing is still very much in it's infancy, and comparing the scale of the numbers involved in RSA vs the scale of numbers demonstrably factorized by Quantum computing so far suggests that this is still very much the case.

5 ElGamal and ECC

This section will touch upon Elliptic Curve Cryptography and ElGamal encryption, two other encryption schemes. This will be done to compare and contrast them with RSA encryption in terms of how they operate and how secure they are. Both of them also use modular arithmetic and group theory in their operation.

5.1 ElGamal

ElGamal is a public key encryption method designed by and named after Taher Elgamal. It operates as follows (A. Menezes 1996):

5.1.1 Key generation

Like in RSA, a large prime number is selected, and modular exponentiation is involved:

1. Generate a large random prime p and a generator α of the multiplicative group $\mathbb{Z}_p^*$ of the integers modulo p (this would effectively be the set of integers $\{1, ...(p-1)\}$)
2. select a random integer n such that $1 \leq n \leq (p-2)$ and compute $\alpha^n \bmod p$
3. the public key is (p, α, α^n) and the private key is n.

A generator is an element n of a cyclic group G such that all elements of G can be represented as powers of n. For example with the group of integers modulo 7, $\{1, 2, 3, 4, 5, 6\}$, the number 3 is a generator:

1. $3^1 \bmod 7 \equiv 3$
2. $3^2 \bmod 7 \equiv 2$

3. $3^3 \bmod 7 \equiv 6$
4. $3^4 \bmod 7 \equiv 4$
5. $3^5 \bmod 7 \equiv 5$
6. $3^6 \bmod 7 \equiv 1$
7. $3^7 \bmod 7 \equiv 3$
8. ...

For a worked example, take the cyclic group U_{17}- it has an order of 16, and it's generators are $3, 10, 5, 11, 14, 7, 12, 6$.

Pick $\alpha = 3$, and pick $n = 10$. For these values, $\alpha^{10} \equiv 8 \bmod 17$.

Publish $(17, 3, 8)$ as a public key, retain 10 as a private key.

5.1.2 Message Encryption

Once the public key is generated and published, it can be used for encryption

1. represent the message as an integer m such that $0 \leq m \leq (p-1)$
2. select a random integer k such that $1 \leq k \leq (p-2)$
3. compute $c_1 \equiv \alpha^k \bmod p$
4. compute $c_2 \equiv m \cdot (\alpha^n)^k \bmod p$
5. send (c_1, c_2) as the ciphertext

Continuing the worked example, with the public key $(17, 3, 8)$ from before, take $m = 7$ and $k = 9$.

Using these values of m and k, calculate $c_1 \equiv \alpha^k \equiv 3^9 \equiv 14 \bmod 17$.

Then calculate $c_2 \equiv m \cdot (\alpha^n)^k \equiv 7 \cdot (8)^9 \equiv 5 \bmod 17$.

5.1.3 Message decryption

Upon receiving the ciphertext (c_1, c_2), it can be decrypted using the private key

1. Use the private key n to calculate $c_1^{p-1-n} \bmod 17$ (Note that $c_1^{p-1-n} \equiv c_1^{-n} \equiv \alpha^{-nk}$).
2. recover m by computing $(c_1^{-n}) \cdot c_2 \bmod p$

Using the example values of $c_1 = 14, c_2 = 5$, calculate c_1^{p-1-n}. This works out to be $14^{17-1-10} = 14^6 \equiv 15 \bmod 17$.

Then recover the original value of m by computing $(c_1^{-n}) \cdot c_2$. With the values calculated earlier, this is $15 \cdot 5 \equiv 7 \bmod 17$.

5.2 Elliptic Curve Cryptography (ECC)

(**NOTE:** figures in the following section were all made manually in Desmos unless stated otherwise. They are used to elaborate on details of ECC that may be difficult to explain using only text)

Elliptic Curve cryptosystems are based on plane curves of the form $y^2 = x^3 + ax + b$ over a finite field, as opposed to the field of real numbers, which is infinite. There is also a requirement that the value of $4a^3 + 27b^2$ be non-zero, and that the polynomial $x^3 + ax + b$ has distinct roots.

ECC is used in Bitcoin to prove ownership of coins ("Elliptic Curve Digital Signature Algorithm" n.d.), is a preferred method of encryption in SSL/TLS, is used to protect US government internal communications, to protect anonymity in Tor browsing, and many other places.

5.2.1 Basic principles of ECC

NOTICE: This section uses O to represent the "point at infinity". This is not the correct symbol used in other documentation for this point, but it has been used due to time constraints. A future revision of this document may contain the correct symbol.

Elliptic Curve Cryptography starts by taking the set of points on a curve E and an additional point "at infinity" O.

An operation of "addition" (represented by the symbol $\oplus$) is defined between two points P and Q on the curve E- this operation involves drawing a line L that crosses the two points and then intersects the curve at a third point R, and then drawing a vertical line L' that passes through R (and is said to pass through the point at infinity O). This Line crosses the curve at a fourth point, defined as $P \oplus Q$. If P and Q are equal, the line L is defined as the tangent to E at P.

By defining E as a curve over a finite field F_q (numbers which follow the laws of a "group" over addition and multiplication) instead of the real numbers, $E(F_q)$ has a finite number of discrete points, as opposed to infinitely many points. To be precise there are $2q + 1$ points- there are q choices for x, and each value of x has at most 2 values for y, and the point at infinity O. These points and the

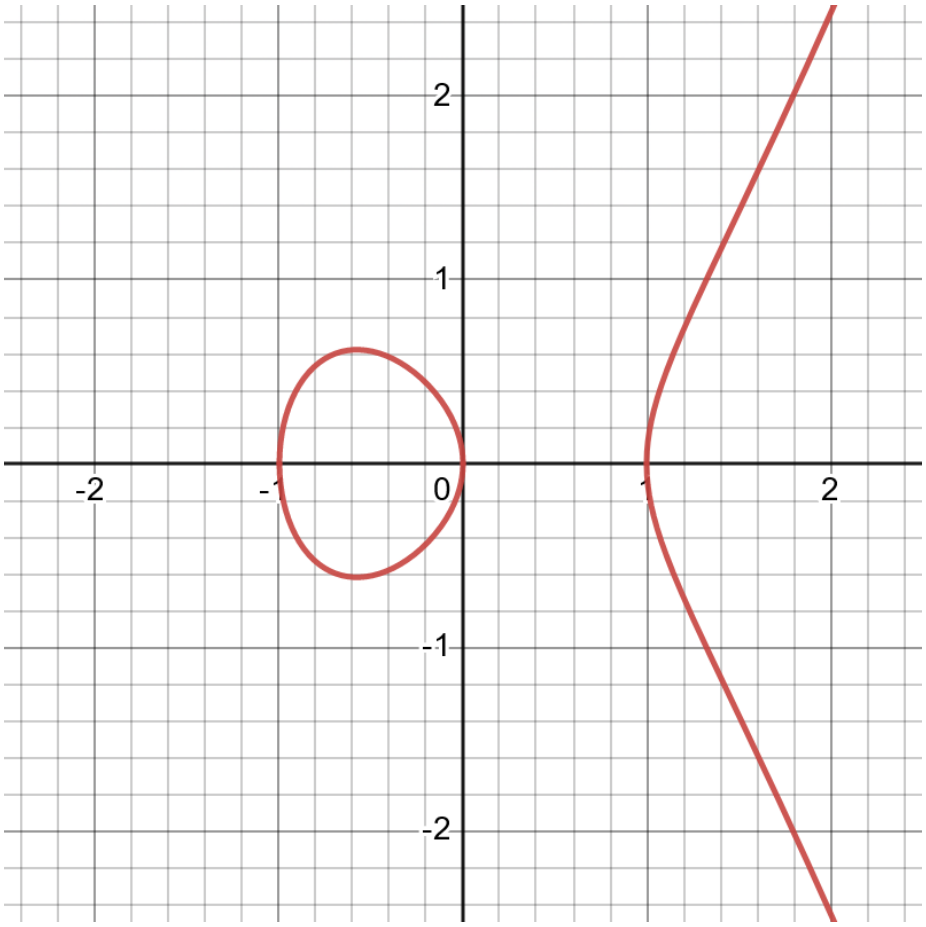

Fig. 1: Figure of the elliptic curve $y^2 = x^3 - x$ generated in Desmos

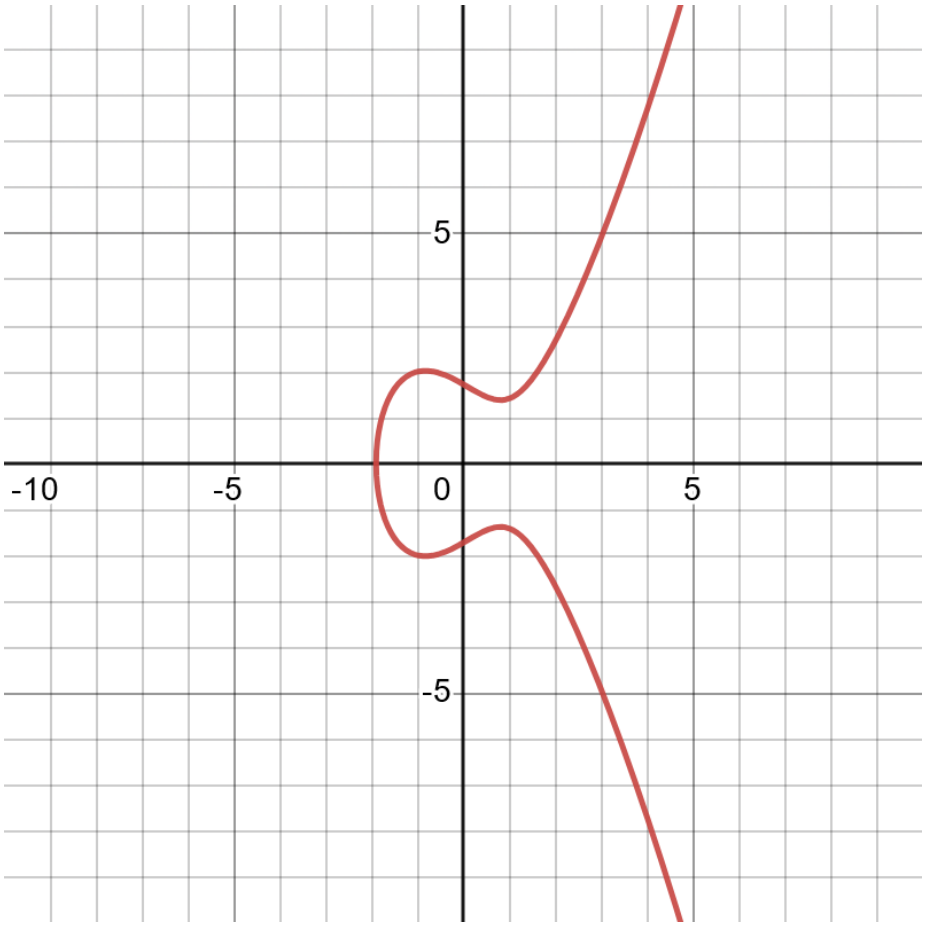

Fig. 2: Figure of the elliptic curve $y^2 = x^3 - 2x + 3$ generated in Desmos

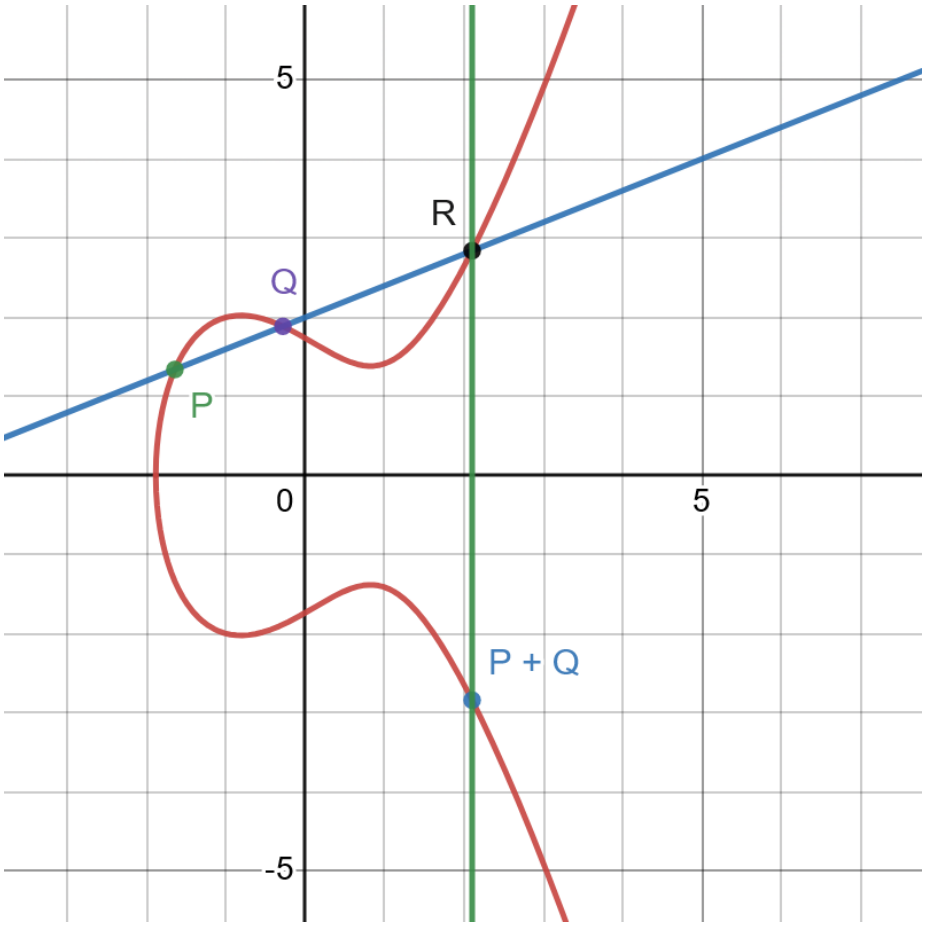

Fig. 3: the operation of "addition" on elliptic curve E (shown in red)- the line L passes through it at points P, Q, R, and the vertical line represents L' and passes through R and $P \oplus Q$ (which can also be considered $-R$), the upper and lower intersections between E and L' respectively. Generated using Desmos

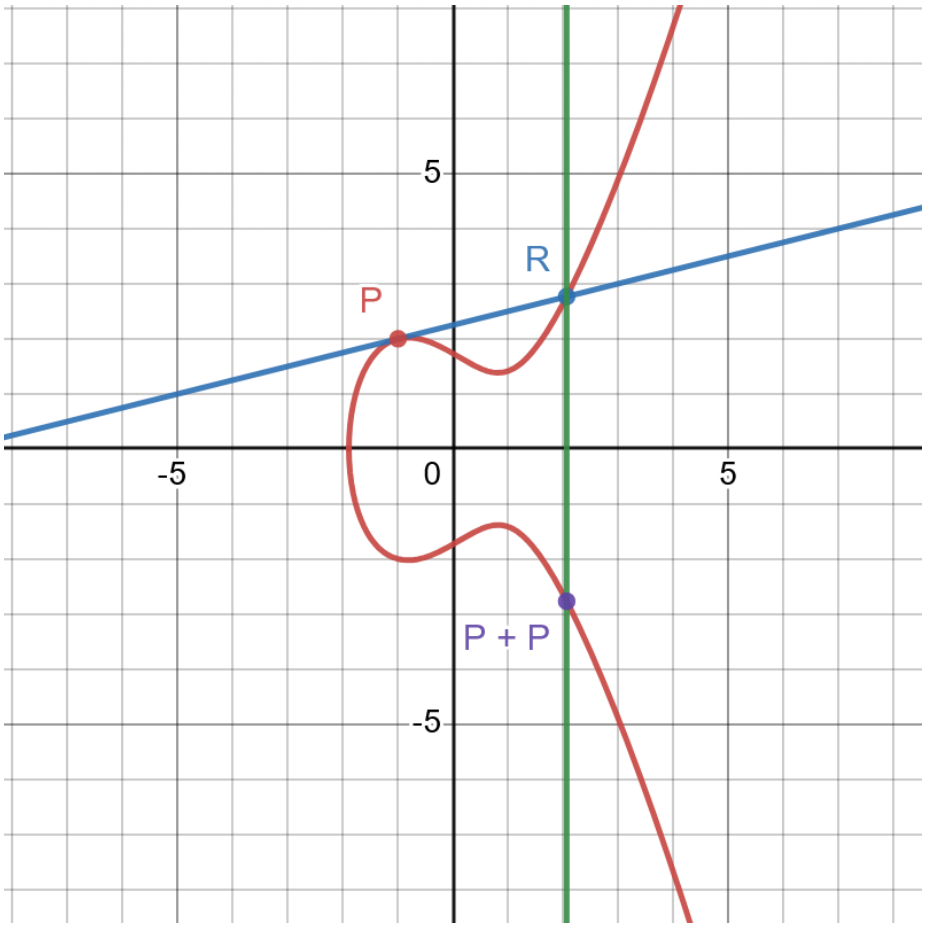

Fig. 4: An elliptic curve $y^2 = x^3 - 2x + 3$ with a tangent line L at a point P. This tangent line crosses the elliptic curve again at a point R. The vertical line through R crosses the curve again at the point $P \oplus P$ (which can also be defined as $-R$). Generated using Desmos

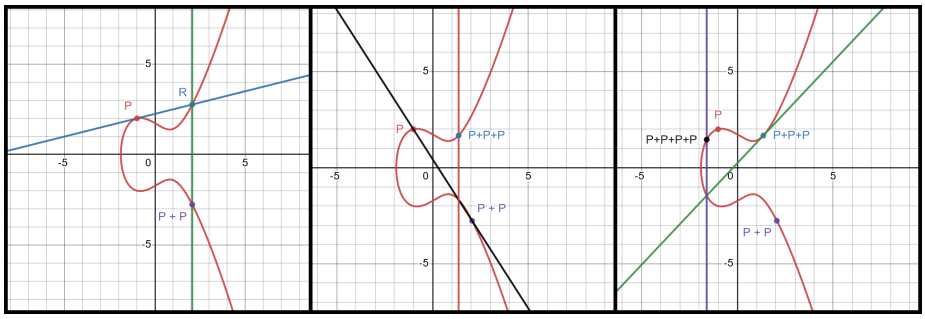

Fig. 5: Repeatedly performing $P \oplus P$ to generate further points ($3P, 4P, 5P...$). Points appear to "jump around". At kP where k is the order of the chosen Field for Elliptic Curve encryption, $kP = P$. Optimizations of this procedure involve taking note of the fact that $P \oplus 2P = 3P$

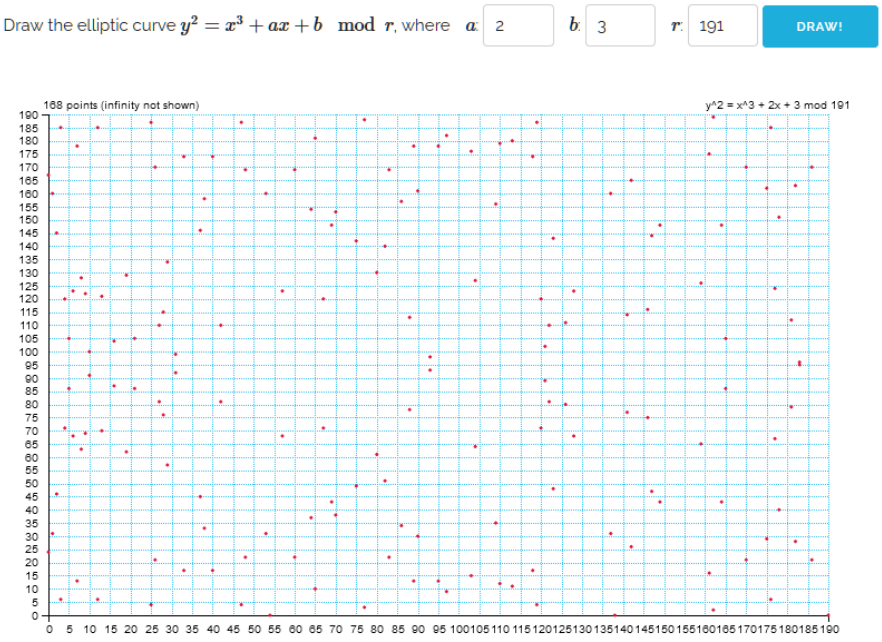

Fig. 6: An Elliptic curve $y^2 = x^3 + 2x + 3$ drawn over the finite field 191 using https://graui.de/code/elliptic2/. Only whole number coordinates of (x, y) are shown, and the curve "wraps around" to produce many of these points

previously defined operation of "addition" form an Abelian group G (the "point at infinity" O acts as the "identity element" for this group)(Shevchuk 2020).

It is possible to perform this "addition" operation very efficiently (especially the tangent case with $P \oplus P$), but it is not easy to perform the reverse of this operation.

5.2.2 Parameter selection

The first part of Elliptic Curve Encryption involves two parties agreeing on parameters p, a, b, P, n, h, where p is a prime number, a, b are parameters that define the elliptic curve E, P is a point on E, n is the "order" of P (the least integer for which $nP = O$, and usually a prime number itself) and h is a cofactor of the group G (the integer such that $nh = |G|$).

In actual practice however, These parameters are not usually generated by the parties themselves as this would be computationally expensive, rather they are instead selected from a set of "standard" or "named" curves published by a standards body such as NIST ("Digital Signature Standard (DSS)" n.d.) or the SEC (Brown n.d.). One such curve endorsed by the NIST (or at least, announced to be endorsed by them in 2017) is Curve25519, which uses a finite field of the prime number $2^{255} - 19$, the curve $y^2 = x^3 + 486662x^2 + x$ and the base point $x = 9$ (Kleppmann n.d.).

There is also an option to use a "binary field" instead of a "prime field"- a number of the form 2^m, which necessitates an "auxiliary curve", called f, thus the parameters are m, f, a, b, P, n, h

5.2.3 Key exchange in ECC

After deciding on the parameters of the curve and the base point, two parties can use ECC to perform Diffie-Hellmann key Exchange-

1. Both parties, A and B each select a key k from the interval $1, ...n - 1$ (where n is the order of the point in the field chosen... for Curve22519 this is a large prime number slightly bigger than 2^{252}).

2. The base point P is exponentiated using a repeated $P \oplus P$ operation (as seen in Figure 6). The resultant point $Q = kP$ is the private key. This results in pairs of keys (k_A, Q_A) and (k_B, Q_B).

3. the parties exchange public keys, and use them to compute $k_A Q_B$ and $k_B Q_A$. These two points will be equivalent, as $k_A Q_B = k_A k_B P$ and $k_B Q_A = k_B k_A P$.

4. this point can now be used for symmetric encryption as a shared secret

5.2.4 ElGamal Encryption using ECC

An analogue to ElGamal encryption can be performed using Elliptic curves (Shevchuk 2020).

1. Select the curve and base point P of order n, and generating the public key Q as described in the previous "key exchange" section
2. use a function f to map a plaintext m to a point M on your chosen curve $f(m) = M$.
3. Choose a random value d from the interval $1...n-1$
4. compute $C = dP$
5. compute $D = M + dQ$
6. the ciphertext is the points (C, D)
7. the recipient computes $M = D - kC$ using their key k, where $kC = k(dP) = d(kP) = dQ$
8. they can then perform $f^{-1}(M) = m$ to retrieve the plaintext.

5.3 Security and other comparisons of ECC, ElGamal, RSA

ECC, RSA and ElGamal all rely on the difficulty of a "discrete logarithm" problem, that is working out the value of$b$ for an expression $a^b = c$ where a, c are elements of a group G. Depending on the way the group is defined, how many elements it has, and how the operation of the group is defined, it can be extremely difficult to solve.

Based on existing literature it can be discerned that all three are at risk of being cracked by Quantum computing if the technology manages to progress to that point.

Certain sources claim that ECC provides comparatively stronger security for a smaller key size (in terms of classical computing), however in 2008 researchers John Proos and Christof Zalka claimed that a small quantum computer would require 1000 qubits to crack a 160-bit Elliptic Curve whereas it would take 2000 qubits to crack a 1024-bit RSA key (J. Proos 2008).

RSA is more widely used than ECC primarily because it is older and thus "more trusted" compared to ECC.

All cryptosystems suffer from security problems if they are not properly implemented

6 Potential research areas

Based on this study of RSA and the brief overview of other related cryptosystems a few potential areas can be identified for further study:

1. To check the prevalence of the vulnerabilities identified by Heninger et al (Nadia Heninger 2012) in TLS traffic in 2023 compared to 2012
2. Potential investigation of the Hastad attack on values of e that are not 3 (possibly 17).
3. While the feasibility of research involving quantum computing is limited, it is noted that there are more papers on researching using quantum computing for cracking RSA and for using Shor's algorithm to crack RSA than there are for using Quantum computing to crack ECC. It may be an interesting avenue to pursue.

7 Research plan and goals

The plan for this study depends on which of the aforementioned "potential research areas" is pursued, however it is very likely that it will involve the development, testing and implementation of likely small programs to demonstrate or investigate traits of these cryptosystems (or of things that use these cryptosystems, such as internet traffic or certificates).

Programs will be written in C and C++ for speed reasons, and developed on both Windows (Windows 10, with Cygwin64) and Unix environments (Lubuntu, with GCC).

Development may switch to primarily being on Unix later to take advantage of the ease of setting up C/C++ libraries and packages compared to Windows.

Linux/Lubuntu environment will also be advantageous/necessary if network traffic and packet analysis is necessary (especially if vulnerability checking path is pursued).

The code for these programs will be maintained on a GitHub repository that will be updated on a regular basis. Presentations may consist of software demos in addition to slides.

Other languages such as Python may be investigated and considered- However bear in mind Python is an interpreted language, and may be slower compared to a compiled language like C/C++.

8 Methodology and Professional issues

If we pursue the path of traffic security analysis, we will likely run into a number of ethical and privacy concerns as this involves examining other people's traffic to see how prevalent insecure certificates still are.

Relevant permits and informed consent will have to be considered.

Even if this specific path is not pursued, this study does touch upon things that

References

Cocks, Clifford (1973). "A note on "Non-secret encryption"". In: URL: https://staff.csie.ncu.edu.tw/yensm/lecture/Cryptography/supplementary/A%5C%20Note%5C%20on%5C%20Non-Secret%5C%20Encryption%5C%20by%5C%20C%5C%20Cocks%5C%20(1973).pdf.

R.L Rivest A. Shamir, L. Adleman (1978). "A Method for Obtaining Digital Signatures and Public Key Cryptosystems". In: *Communications of the ACM Volume 21.*

Yao, A. (1982). "Theory and applications of trapdoor functions". In: URL: https://www.di.ens.fr/users/phan/secuproofs/yao82.pdf.

M. Bellare, P. Rogaway (1995). "Optimal Asymmetric Encryption- How to Encrypt with RSA". In: *Advances in Cryptology - Eurocrypt '94 Proceedings, Lecture Notes in Computer Science Vol. 950, A. De Santis ed, Springer-Verlag, 1995.* URL: https://cseweb.ucsd.edu/~mihir/papers/oaep.pdf.

A. Menezes P. van Oorschot, S. Vanstone (1996). *Handbook of Applied Cryptography.* CRC Press.

Kaliski, Burt (1998). *PKCS #1: RSA Encryption Version 1.5.* DOI: 10.17487/RFC2313. URL: https://www.rfc-editor.org/info/rfc2313.

Schneier, B. (2003). *Applied Cryptography- Protocols, Algorithms and Source Code in C.* Wiley Books.

J. Proos, C. Zalka (2008). "Shor's discrete logarithm quantum algorithm for elliptic curves". In: URL: https://arxiv.org/pdf/quant-ph/0301141.pdf.

Xin Zhou, Xiaofei Tang (2011). "Research and Implementation of RSA Algorithm for Encryption and Decryption". In.

Nadia Heninger Zakir Durumeric, Eric Wustrow (2012). "Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Device". In: URL: https://factorable.net/weakkeys12.conference.pdf.

Anh Do Soe Thet Ko, Aung Thu Htet (2013). *ELECTROMAGNETIC SIDE-CHANNEL ANALYSIS ON INTEL ATOM PROCESSOR.*

Chennagiri, A. (2017). "A Tutorial paper on Hastad Broadcast Attack". In: URL: http://koclab.cs.ucsb.edu/teaching/cren/project/2017/chennagiri.pdf.

J. Knockel T. Ristenpart, J. Crandall (2018). "When Textbook RSA is Used to Protect the Privacy of Hundreds of Millions of Users". In: URL: https://arxiv.org/pdf/1802.03367.pdf.

"Seriously, stop using RSA" (2019). In: URL: https://blog.trailofbits.com/2019/07/08/fuck-rsa/.

F. Boudot N. Heninger, E. Thomé (2020). "Comparing the Difficulty of Factorization and Discrete Logarithm: A 240-Digit Experiment". In: DOI: 10.1007/978-3-030-56880-1_3.

Shevchuk, Olga (2020). "INTRODUCTION TO ELLIPTIC CURVE CRYPTOGRAPHY". In: URL: https://math.uchicago.edu/~may/REU2020/REUPapers/Shevchuk.pdf.

Nikesh S. Dattani, Nathaniel Bryans (2021). "Quantum factorization of 56153 with only 4 qubits". In: DOI: https://doi.org/10.48550/arXiv.1411.6758.

(N.d.). In: (). URL: https://web.archive.org/web/19971210213248/http://lavarand.sgi.com/.

Brown, Daniel (n.d.). "SEC 2: Recommended Elliptic Curve Domain Parameters". In: (). URL: http://www.secg.org/sec2-v2.pdf.

"Digital Signature Standard (DSS)" (n.d.). In: (). URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

"Elliptic Curve Digital Signature Algorithm" (n.d.). In: (). URL: https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm.

"Fundamental types" (n.d.). In: (). URL: https://en.cppreference.com/w/cpp/language/types.

Kleppmann, M. (n.d.). "Implementing Curve25519/X25519: A Tutorial on Elliptic Curve Cryptography". In: (). URL: https://www.cl.cam.ac.uk/teaching/2122/Crypto/curve25519.pdf.

R. Selvarajan V. Dixit, X. Cui (n.d.). "Prime factorization using quantum variational imaginary time evolution". In: (). DOI: https://doi.org/10.1038/s41598-021-00339-x.

"The Extended Euclidean Algorithm" (n.d.). In: (). URL: https://www.extendedeuclideanalgorithm.com/xea.php.