Computational Number Theory

Comparative Analysis of RSA and Elliptic Curve Cryptography

Efficiency, Key Size, and Security Protocols

Sohan Suchdev

Abstract

This paper contrasts the mathematical foundations and computational efficiency of RSA encryption against Elliptic Curve Cryptography. The study analyses the discrete logarithm problem versus integer factorisation, evaluating performance metrics such as key generation speed and encryption overhead. Results highlight the advantages of ECC in resource-constrained environments. A look towards the future about quantum computing, looking at the implication of Shor's algorithm on modern encryption techniques.

Word Count: 4000

Contents

1 Introduction

Encryption involves encoding information by converting plaintext into a coded ciphertext ensuring only authorised parties can read the information [1].

There are two primary encryption types: symmetric and asymmetric.[1] Symmetric encryption uses the same key for encryption and decryption. This is generally faster as it is simpler, but requires secure key sharing between parties. Asymmetric encryption uses a pair of keys: a public key which is accessible to anyone, used for encryption, and a private key, kept confidential and used for decryption. This is more secure as the private key is never shared.

This essay explored the mathematical foundations of two asymmetric encryption methods: RSA and Elliptic Curve Cryptography (ECC). In both cases, a public key is shared with anyone sending a secure message, and a private key just the recipient. The sender encrypts the message using the recipient's public key, generating ciphertext which is sent to the recipient, using their private key to decrypt into plaintext.

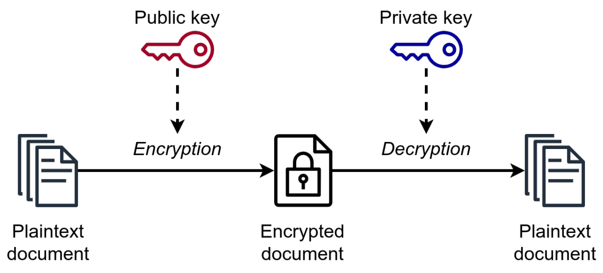


Figure 1: *Asymmetric Encryption [2]*

RSA is based on the difficulty of factoring large prime numbers[3], while ECC relies on elliptic curves over finite fields[4]. Recent research suggests ECC may offer advantages over RSA, such as speed and security[5].

For RSA, primality and its role in encryption will be explored. The discrete logarithm problem will be examined, facilitating one-way trapdoor function creation. Additionally, Euler's totient function and Euler's theorem, fundamental in deriving the RSA encryption and decryption processes, will be discussed.

For ECC, the structure of elliptic curves within finite fields will be explored, covering group laws and point operations, which form the backbone of ECC encryption[6]. The Elliptic Curve Discrete Logarithm Problem (ECDLP) will be discussed, another critical one-way trapdoor function underpinning ECC's security..

The algorithms will be compared in terms of security and time complexity (using Big O notation) before considering the impacts of new technologies and threats.

2 Modular Arithmetic

Modular arithmetic represents numbers with their proportion to a larger number, called the modulus, after which the count is reset. For example, in a 24-hour clock ($N = 24$), the count resets to 0 at midnight[7]. This is useful for applications such as encryption.
If a and b are integers, and n a positive integer, then a is congruent to b modulo n if a and b have the same remainder when divided by n. This relationship is written as[7]:

$$a \equiv b \pmod{n}$$

Formally, using | to denote division,

$$a \equiv b \pmod{n} \iff n \mid (a-b) \iff kn = (a-b), \quad k \in \mathbb{Z}.$$

For example, 17 and 5 are congruent in modulus 12 because:

$$17 - 5 = 12$$

$$\frac{12}{12} = 1$$

$$\therefore 17 \equiv 5 \pmod{12}$$

2.1 Properties of Modular Arithmetic: Symmetry and Reflexivity

Symmetry: If $a \equiv b \pmod{n}$, then $b \equiv a \pmod{n}$: congruence is a symmetric relation. For example, if $17 \equiv 5 \pmod{12}$, then $5 \equiv 17 \pmod{12}$. This symmetry arises because the relation is defined by the difference $a - b = kn$, where $k \in \mathbb{Z}$, which is equivalent to $b - a = -kn$.
Reflexivity: Any integer is congruent to itself modulo n: $a \equiv a \pmod{n}$ for any integer a and positive integer n. This is because $a - a = 0$, and n always divides 0. For example, $9 \equiv 9 \pmod{7}$.

3 RSA Encryption

RSA encryption operates as a "trapdoor one-way function" meaning encryption is easy but decryption is nearly impossible without the private key (the "trapdoor"). RSA relies on the difficulty of factoring the product of two large primes making decryption difficult without the specific key. It does this by the discrete logarithm problem[8].

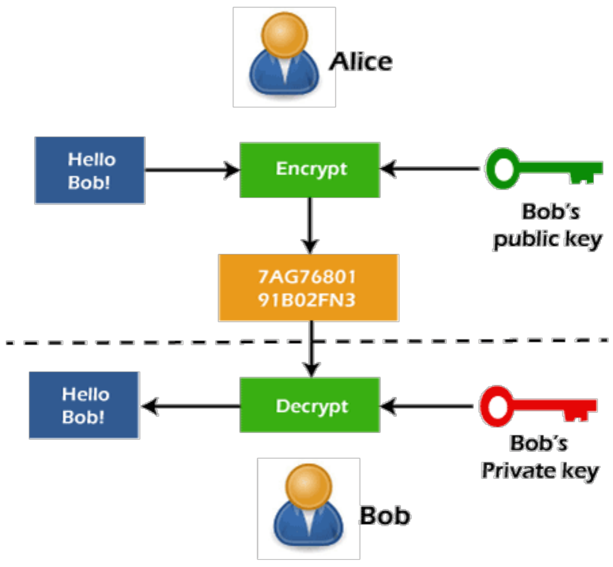


Figure 2: *RSA encryption [9]*

3.1 Primality

A prime number is a natural number greater than 1 with two distinct positive integer factors, 1 and itself[10]. So, if p is prime:

$$p \in \mathbb{N} \text{ and } p > 1 \text{ such that for all } d \in \mathbb{N}, \text{ if } d \text{ divides } p \text{ then } d = 1 \text{ or } d = p.$$

Encryption often depends on the size of the chosen prime numbers, increasing the computational complexity of breaking encryption[11]. If the set of primes were finite, this complexity would be limited, and encryption would eventually become vulnerable to advancements in computational power[12]. So it is important to consider whether the set of primes is infinite.

Euclid's Proof of infinite primes

Assume a finite number of primes, $p_1, p_2, p_3, \ldots, p_n$. Define N as:

$$N = p_1 p_2 p_3 \cdots p_n + 1$$

Since $N > 1$, it must have at least one prime divisor. However, none of the primes p_i (where $i = 1, 2, \ldots, n$) can divide N without leaving remainder 1. Thus, N is not divisible by any of $p_1, p_2, \ldots, p_n$. This contradicts finite primes, as N must have a prime divisor that is not among $p_1, p_2, \ldots, p_n$. Therefore, there must be infinitely many primes[13].
As there are infinite primes, the complexity scales indefinitely.

3.1.1 Greatest Common Divisor and Euclidean Algorithm

The greatest common divisor gcd of a and b, where $a, b \neq 0$, is the greatest positive integer that divides both a and b, denoted $\gcd(a, b)$.

The Euclidean algorithm calculates the greatest common divisor of two integers[14].

The Euclidean Algorithm

The Euclidean algorithm works because if a number d divides both a and b, then d must also divide the remainder r when a is divided by b. Therefore, the $\gcd(a, b)$ is equivalent to $\gcd(b, r)$, resulting in the following algorithm:

1. Start with two positive integers a and b, where $a > b$.
2. Divide a by b and obtain remainder r:

$$a = bq + r$$

where q is the quotient and r the remainder.

3. If $r \neq 0$, replace a with b, and b with r
4. Repeat until remainder r is 0. The final non-zero remainder is the $\gcd(a, b)$.

Example: Finding $\gcd(252, 105)$

Step	**Operation**	**Values**
1	Start with two integers $a = 252$ and $b = 105$	$a = 252, b = 105$
2	Divide $a = 252$ by $b = 105$, find remainder r	$252 = 105 \cdot 2 + 42$ $r = 42$
3	Replace a with $b = 105$ and b with $r = 42$	$a = 105, b = 42$
2	Divide $a = 105$ by $b = 42$, find remainder r	$105 = 42 \cdot 2 + 21$ $r = 21$
3	Replace a with $b = 42$ and b with $r = 21$	$a = 42, b = 21$
2	Divide $a = 42$ by $b = 21$, find remainder r	$42 = 21 \cdot 2 + 0$ $r = 0$
4	The remainder is 0. The last non-zero remainder is the gcd	$\gcd(252, 105) = 21$

Overall, the algorithm can be summarised as[14]:

$$\gcd(a, b) = \gcd(b, r)$$

In encryption, this is useful for ensuring the public key is valid and finding the private key[14].

3.1.2 Co-Primality

Two integers a, b are co-prime if[15]:

$$\gcd(a, b) = 1$$

$$e.g \ \gcd(3, 5) = 1$$

3.1.3 Composite Numbers

A composite number is a positive integer greater than 1 that is not prime[16]. A positive integer n is composite if there exist integers a and b such that:

$$1 < a < n \quad \text{and} \quad 1 < b < n \quad \text{and} \quad a \cdot b = n$$

Composite numbers are important because their factorisation difficulty ensures secure encryption[12].

3.2 The Discrete Logarithm Problem(DLP)

The discrete logarithm is the inverse of modular exponentiation, much like the classical logarithm is the inverse of regular exponentiation[10]. The term “discrete” highlights that the inputs are integers within a finite set, rather than continuous real numbers. While modular exponentiation is easy to compute, finding the discrete logarithm is computationally hard, so the DLP is fundamental to RSA encryption’s security.

The problem asks, given integers a, b, and n where:

$$b \equiv a^x \pmod{n}$$

find an integer x (the discrete logarithm) such that the above equation holds[17]. Considering an example to understand how the DLP works.

- Let $a = 3$ and $n = 17$.
- **Then**:

$$
\begin{aligned}
&3^1 = 3 \equiv 3 \pmod{17}\\
&3^2 = 9 \equiv 9 \pmod{17}\\
&3^3 = 27 \equiv 10 \pmod{17}\\
&3^4 = 81 \equiv 13 \pmod{17}\\
&3^5 = 243 \equiv 5 \pmod{17}\\
&3^6 = 729 \equiv 16 \pmod{17}\\
&3^7 = 2187 \equiv 14 \pmod{17}\\
&3^8 = 6561 \equiv 8 \pmod{17}\\
&3^9 = 19683 \equiv 7 \pmod{17}\\
&3^{10} = 59049 \equiv 4 \pmod{17}\\
&3^{11} = 177147 \equiv 12 \pmod{17}\\
&3^{12} = 531441 \equiv 2 \pmod{17}\\
&3^{13} = 1594323 \equiv 6 \pmod{17}\\
&3^{14} = 4782969 \equiv 1 \pmod{17}
\end{aligned}
$$

For instance, if $b = 7$ then it is known from the above that $x = 9$ because $3^9 \equiv 7 \pmod{17}$" While it is easy to compute $a^x \pmod{n}$ if x is known, finding x given a, b, and n is hard, requiring trial and error. This asymmetry makes DLP useful in cryptography, being impractical for large numbers, creating the "one-way function" for RSA encryption. So a private key that needs to be solved with the DLP is effective.

3.3 Euler's Totient Function

Euler's totient function, $\phi(n)$, helps find the value of a private key by calculating the number of integers 1 to n that are coprime with n[18]:

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right), n \in \mathbb{Z}^+ \tag{1}$$

where the product is taken over all distinct primes $p \mid n$.

The term $\left(1 - \frac{1}{p}\right)$ represents the fraction of integers not divisible by each prime factor p of n; for instance, if $p = 11$, then $\frac{10}{11}$ of the integers up to n are coprime to 11. The formula multiplies these exclusion fractions for all distinct prime factors, and then multiplying n by this product of fractions, results in $\phi(n)$.

Examples: Finding $\phi(30)$

Step	Calculation	Result
1	Identify prime factors of 30: $p_1 = 2$, $p_2 = 3$, $p_3 = 5$	-
2	Calculate $\left(1 - \frac{1}{p_1}\right)$: $\left(1 - \frac{1}{2}\right)$	$\frac{1}{2}$
3	Calculate $\left(1 - \frac{1}{p_2}\right)$: $\left(1 - \frac{1}{3}\right)$	$\frac{2}{3}$
4	Calculate $\left(1 - \frac{1}{p_3}\right)$: $\left(1 - \frac{1}{5}\right)$	$\frac{4}{5}$
5	Multiply all fractions together: $\prod_{p\|30} \left(1 - \frac{1}{p}\right) = \frac{1}{2} \times \frac{2}{3} \times \frac{4}{5}$	$\frac{4}{15}$
6	Calculate $n \cdot \prod_{p\|n} \left(1 - \frac{1}{p}\right)$: $30 \times \frac{4}{15}$	8

3.3.1 Multiplicative Properties

Exploring the multiplicative properties of Euler's totient function is important, as it allows for quicker private key creation[18]. To explore the multiplicative relationship, sets will be used. A set is a collection of distinct objects(numbers, symbols, or other mathematical structures) where each object appears only once, and the set is treated as an individual object[10]. Sets are useful for exploring Euler's totient function through establishing a one-to-one correspondence among coprime integers.

To begin,

$$A = \{k \in \mathbb{Z} \mid 1 \leq k \leq a,\ \gcd(k, a) = 1\}$$

A is the set of integers from 1 to a that are coprime to k and $|A|$, the cardinality of a set, is the number of objects within the set. Here, the number of integers in A is $\phi(a)$, so $|A| = \phi(a)$.

Similarly:

$$B = \{l \in \mathbb{Z} \mid 1 \leq l \leq b,\ \gcd(l, b) = 1\}$$

$|B| = \phi(b)$.

So, the set of integers from 1 to ab that are coprime to ab is:

$$C = \{m \in \mathbb{Z} \mid 1 \leq m \leq ab,\ \gcd(m, ab) = 1\}$$

$|C| = \phi(ab)$.

There is a one-to-one correspondence (each element of one set maps to exactly one element of the other due to m and n's co-primality) between pairs (k, l) where $k \in A$ and $l \in B$, and the elements of C[10]. This means each pair (k, l) uniquely corresponds to an element in C, and $|C|$ is simply the product of $|A|$ and $|B|$[10].
Thus:

$$|C| = |A| \times |B|$$

$$\therefore |C| = \phi(a) \times \phi(b)$$

$$\therefore \phi(ab) = \phi(a) \cdot \phi(b)$$

So, Euler's totient function is multiplicative, making private key creation easier.

3.3.2 Prime Numbers

For a prime p, the only prime factor is p, so Euler's totient function can be simplified using (1):

$$\phi(p) = p\prod_{q|p}\left(1 - \frac{1}{q}\right) = p\left(1 - \frac{1}{p}\right)$$

$$\therefore \phi(p) = p\left(1 - \frac{1}{p}\right) = p \cdot \frac{p-1}{p} = p - 1$$

Thus, for a prime number p:

$$\phi(p) = p - 1$$

Therefore, using the multiplicative properties and prime number properties of $\phi(n)$, if n can be split up into its prime factorisation, which is computationally demanding, it can be straightforward to calculate $\phi(n)$. *E.G*:

$$n = pq \text{ where } p \text{ and } q \text{ are prime numbers}$$

$$\phi(n) = \phi(pq)$$

$$\phi(pq) = \phi(p)\phi(q)$$

$$\phi(pq) = (p-1)(q-1)$$

3.4 Euler's Theorem

Euler's totient function helps calculate the private key but needs to work with the DLP and modular arithmetic to create a secure private key[3]. To achieve this, the properties of sets of integers that are coprime to a given modulus will be explored, facilitating the use of RSA.

Consider the set of integers $R = \{1, 2, \ldots, n\}$ and remove those not coprime with n. This is a reduced residue[16], with the following properties[14]:

- $\forall r \in R,\ \gcd(r, n) = 1$

- $|R| = \phi(n)$
- $\forall a, b \in R,\ a \not\equiv b \pmod{n}$

Let $S = \{x_1, x_2, \ldots, x_{\phi(n)}\}$ be the set of reduced residues modulo n. Multiplying each element of S by a, where $\gcd(a, n) = 1$ and $a \in \mathbb{Z}$, gives another set of integers $T = \{ax_1, ax_2, \ldots, ax_{\phi(n)}\}$, which is also a set of reduced residues modulo n.

Each element of T is distinct modulo n because multiplying each element by a, which is coprime with n, only permutes the elements of S; *i.e.* elements in T cover the same values as those in S, just in a different order.

Therefore, $T \equiv S \pmod{n}$.

So:

$$x_1 x_2 \cdots x_{\phi(n)} \equiv (ax_1)(ax_2) \cdots (ax_{\phi(n)}) \pmod{n}$$

$$x_1 x_2 \cdots x_{\phi(n)} \equiv a^{\phi(n)} x_1 x_2 \cdots x_{\phi(n)} \pmod{n}$$

Since $\gcd(x_1 x_2 \cdots x_{\phi(n)}, n) = 1$:

$$1 \equiv a^{\phi(n)} \pmod{n}$$

This is known as Euler's theorem, stating that if n is a positive integer and a is an integer $\gcd(a, n) = 1$[14] then:

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

3.5 The Process

It is now possible to encrypt. First, a number m (the message) is taken, raised to some exponent e (the public key), and divided by the modulus N (the public key) to get the remainder c (ciphertext).

$$c = m^e \pmod{N}$$

To decrypt, it is necessary to raise c to another exponent d, to return to m. Combining both:

$$m = c^d \pmod N$$

$$\therefore m = m^{ed} \pmod N$$

Euler's Theorem can be used to determine d.

$$m^{\phi(N)} \equiv 1 \pmod N$$

This can be modified to find d by raising both sides to some integer k. The value of k represents any integer that helps scale the equation while preserving its congruence[13] but must be chosen carefully for the relationship to hold[2]. The congruence remains, so this can be written as:

$$m^{k\cdot\phi(N)} \equiv 1 \pmod N$$

By the properties of modular arithmetic, it is possible to multiply both sides by m, maintaining the congruence, thus:

$$m^{k\cdot\phi(N)+1} \equiv m \pmod N$$

It is known that $k \cdot \phi(N) + 1 = e \times d$, which can be rewritten to solve for d. Therefore, the private key has been created.

$$d = \frac{k \cdot \phi(N) + 1}{e}$$

3.5.1 Key Generation

To generate the keys, choose two large, random prime numbers p and q, ensuring both are sufficiently large and significantly different to maximise security. This large difference between p and q reduces the likelihood of certain attacks, such as Fermat's factorization method, which is more effective when p and q are close in size [19]. A standard method is to select random integers and use a primality test until two primes are found[20]. In this example, small values are chosen for p and q, but in reality, these numbers should have hundreds of digits[3].

p and q are kept secret.

$$\text{e.g. } p = 59, \quad q = 53$$

Calculate $N = pq$. Its length, usually expressed in bits, is the key length.

$$\text{e.g. } N = pq = 59 \times 53 = 3127$$

Calculate $\phi(N)$ where $\phi(N) = \phi(pq) = (p-1)(q-1)$, which makes it very quick. Keep $\phi(N)$ secret.

$$\text{e.g. } \phi(N) = \phi(pq) = (p-1)(q-1) = (59-1)(53-1) = 58 \times 52 = 3016$$

Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$. The most commonly chosen value for e is 65537. Although choosing $e = 3$ allows for faster encryption, it is less secure due to vulnerabilities like small exponent attacks [19]; however, for simplicity in this example, $e = 3$ is used, and it is released as part of the public key.

$$\text{e.g. } e = 3$$

Determine d, which is the private key, by using the derived formula and setting $k = 2$. d is kept secret as the private key exponent.

$$\text{e.g. } d = \frac{2 \cdot 3016 + 1}{3} = 2011$$

3.5.2 Encryption

The public key is sent to the user, who then can now encrypt their message, $m = 89$, using:

$$c = m^e \pmod N$$

$$e.g\ c = 89^3 \pmod{3127} \equiv 1394 \pmod{3127}$$

3.5.3 Decryption

The ciphertext, $c = 1394$, is sent back and the recipient can decrypt it using the private key:

$$m = c^d \pmod{N}$$

$$e.g\ m = 1394^{2011} \pmod{3127} \equiv 89 \pmod{3127}$$

3.6 Primality tests

RSA encryption relies on very large prime numbers of its keys. The difficulty lies in factoring the modulus N, which is the product of two large primes p and q. Finding these factors is computationally intensive since no formula can list all primes below a given N, and forms the basis of RSA's strength. Consequently, primality tests, such as the Miller-Rabin test and Fermat's test, are employed[21]. Although these tests will not be discussed in this essay, it is important to note that they play a crucial role in ensuring the security of RSA[21].

4 Elliptic Curve Cryptography (ECC)

4.1 Elliptic curves

An elliptic curve is defined by an equation of the form[22]:

$$y^2 = x^3 + ax + b$$

where $a, b \in \mathbb{R}$.

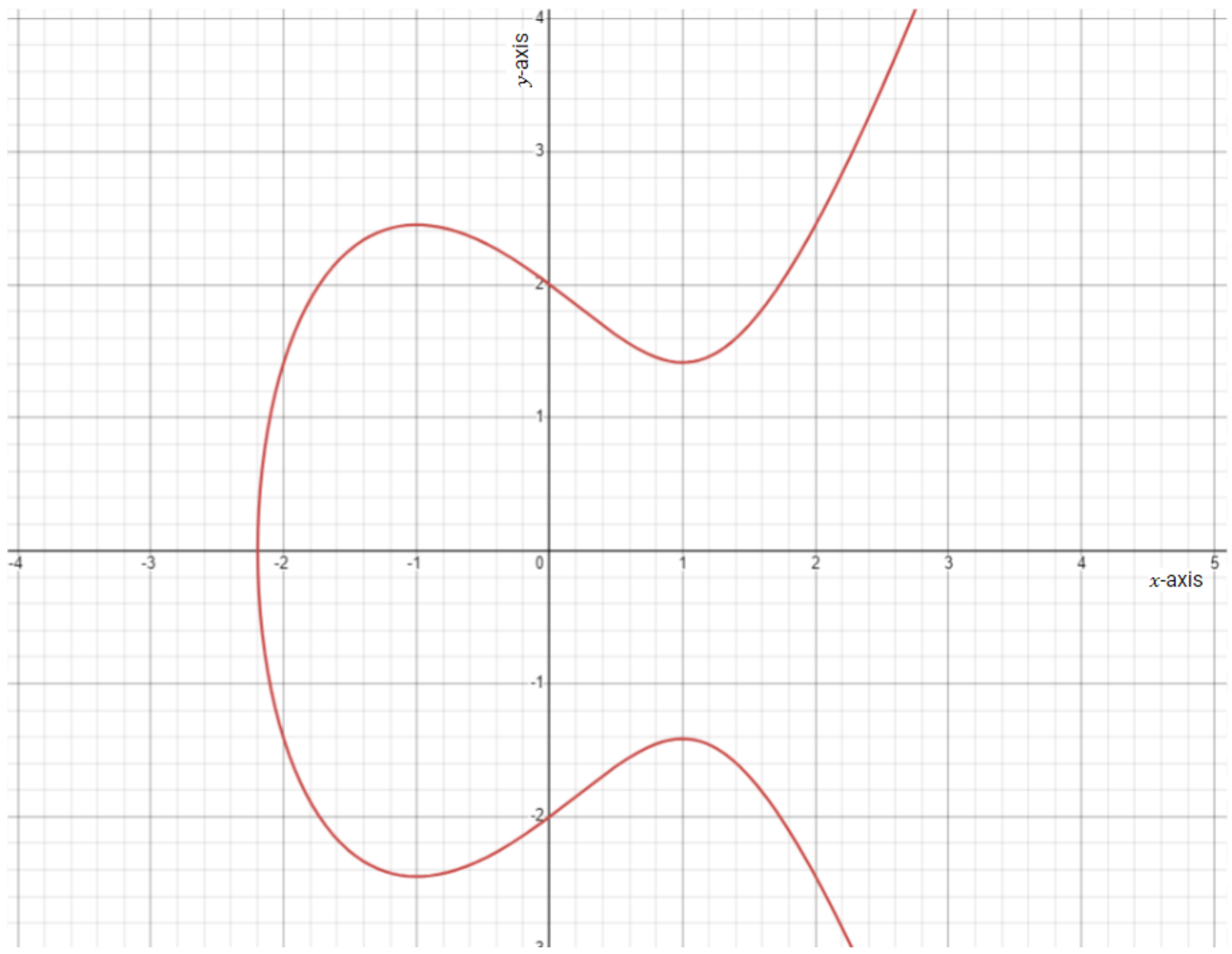


Figure 3: *Eliptical Curve*

4.1.1 Fields

A field is a mathematical structure permitting addition, subtraction, multiplication, and division (except by zero)[23].

A finite field $\mathbb{F}_p$ is a set of q elements where q is a prime power **i.e** $q = p^k$ for some prime p and integer $k \geq 1$). Arithmetic operations within this field are performed modulo p[23].

The elliptic curve can be defined in the finite field:

$$y^2 = x^3 + ax + b \pmod{p} \quad \text{for all } a, b \in \mathbb{F}_p$$

Here y, x, a, b are all within $\mathbb{F}_p$, and the elliptic curve over the finite field $\mathbb{F}_p$ consists of a set of integer coordinates $\{x, y\}$, such that $0 \leq x, y < p$, and stays on the elliptic curve: $y^2 \equiv x^3 + ax + b \pmod{p}$.

Example of elliptic curve over the finite field $\mathbb{F}_{17}$:

$$y^2 \equiv x^3 + 7 \pmod{17}$$

This is illustrated in Figure 4:

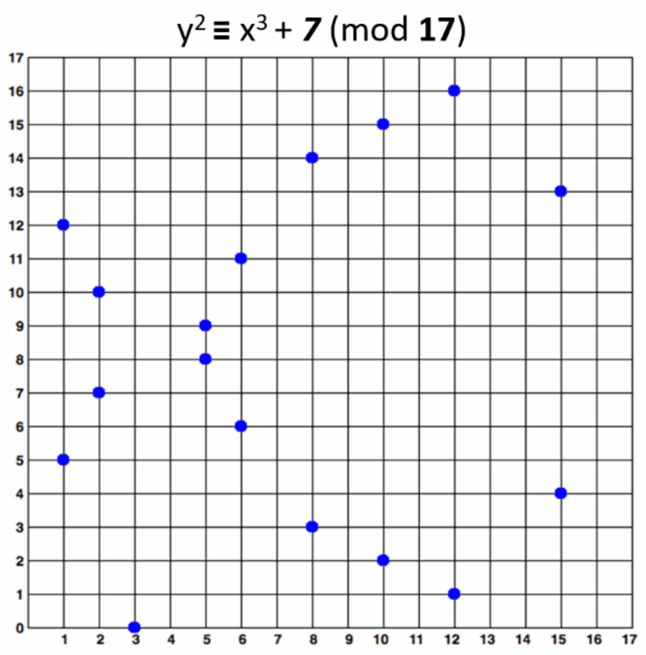


Figure 4: *Eliptic Curve in a finite field [24]*

As shown below, the point $P\{5,8\}$ belongs to the curve, because $(5^3 + 7 - 8^2) \mod 17 = 0$, and the point $\{9,15\}$ does not belong to the curve, because $(9^3 + 7 - 15^2) \mod 17 \neq 0$.

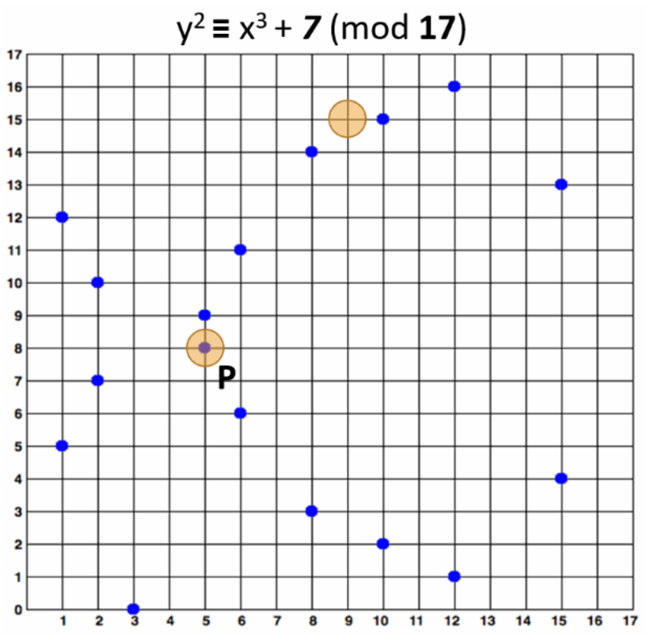


Figure 5: *Eliptic Curve in a finite field [24]*

4.1.2 Singularity case

A singularity on an elliptic curve occurs at points where the curve fails to be smooth[25]. An elliptic curve should not have any singularities to be useful in cryptography, needing to follow the group laws and properties.

In Figure 6, the purple line shows the graph of $y = x^3 + ax + b$ and the blue line $y^2 = x^3 + ax + b$. The value of $x^3 + ax + b$ must be positive as $y^2 \geq 0$. If the purple line is negative, there is no blue curve. When there is a blue curve, it will include both the positive and negative values of the square root.

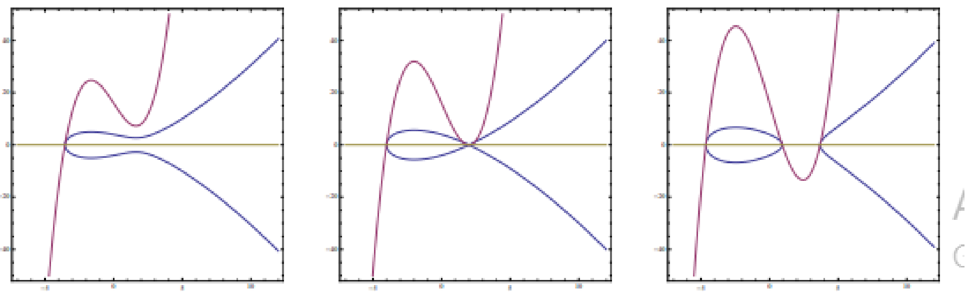


Figure 6: *Elliptic curves with and without singularity [26]*

So, there is a singularity when the purple line is tangential to the x-axis, as the elliptic curve is no longer smooth. Using this a singularity condition can be derived.

When the purple line is tangent to the x-axis, $\frac{dy}{dx} = 0$

$$y = x^3 + ax + b \tag{2}$$

$$\frac{dy}{dx} = 3x^2 + a = 0$$

$$x = \sqrt{-\frac{a}{3}}$$

Knowing the x value for the point, substitute into (2) and simplify:

$$\left(\sqrt{-\frac{a}{3}}\right)^3 + \left(\sqrt{-\frac{a}{3}}\right) a + b = 0$$

$$\left(\sqrt{-\frac{a}{3}}\right)\left(-\frac{a}{3} + a\right) = -b$$

$$\left(\sqrt{-\frac{a}{3}}\right)\left(\frac{2a}{3}\right) = -b$$

$$-\frac{4a^3}{27} = b^2$$

$$4a^3 + 27b = 0$$

This is the discriminant for our curve, and to prevent singularities, it should not be equal to 0. So, the curve

$$y^2 = x^3 + ax + b$$

is non-singular if the discriminant Δ is non-zero:

$$\Delta = -4a^3 + 27b^2 \neq 0$$

4.2 Group Laws and operations

4.2.1 Group Laws

Elliptic curves form an abelian group, where group elements describe the points on the curve along with a point at infinity[22].
The point at infinity, $\mathcal{O}$, acts as the identity element[27]. One understanding of the point at infinity is that, for any point P on the curve, $P + \mathcal{O} = \mathcal{O} + P = P$.

An abelian group is a set G with a binary operation $\circ$[27]:

$$\circ : G \times G \to G$$

1. **Associativity**: $\forall\ x, y, z \in G$,

$$(x \circ y) \circ z = x \circ (y \circ z)$$

2. **Identity**: $\exists$ (there exists an element) $e \in G$ such that $\forall\ x \in G$,

$$e \circ x = x \circ e = x$$

   In ECC, this identity element is $\mathcal{O}$.

3. **Inverse**: $\forall\ x \in G$, $\exists\ y \in G$ such that

$$x \circ y = e = y \circ x$$

   In ECC, the inverse of a point P is $-P$.

4. **Closure**: $\forall\ x, y \in G$,

$$x \circ y \in G$$

5. **Commutativity**: $\forall\ x, y \in G$,

$$x \circ y = y \circ x$$

These group properties are crucial for secure and reliable operations[6], ensuring that point addition behaves predictably, allowing efficient encryption and decryption. They also underpin the security of ECC by making the elliptic curve discrete logarithm problem (ECDLP) difficult to solve, which is key to ECC's strength in cryptography[4].
The point at infinity does not have an actual set of coordinates, and proof of its existence falls beyond the scope of this essay, as do further details of Groups.

4.2.2 Group operations

Point addition involves adding two distinct points P and Q on an elliptic curve E. When adding P and Q, a line is drawn through these points and intersects the elliptic curve at a third point. Reflecting this point across the x-axis gives the result of the addition[24][22].
Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$. The slope m of the line passing through P and Q is:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$e.g\ P = (2, 3), Q = (5, 1) \text{ on an elliptic curve } y^2 = x^3 + 2x + 3.$$

$$e.g\ m = \frac{1-3}{5-2} = \frac{-2}{3} = -\frac{2}{3}$$

The coordinates of the resulting point $R = P + Q = (x_3, y_3)$ are:

$$x_3 = m^2 - x_1 - x_2$$

$$y_3 = m(x_1 - x_3) - y_1$$

$$e.g\ x_3 = \left(-\frac{2}{3}\right)^2 - 2 - 5 = -\frac{59}{9}$$

$$e.g\ y_3 = -\frac{2}{3}(2 + \frac{59}{9}) - 3 = -\frac{154}{27} - 3$$

Also, based on the group laws defined earlier, the following holds:

- $P + Q = Q + P$
- **Associativity**: $P + (Q + R) = (P + Q) + R$

Special Cases

- **Vertical Line**: If $P = (x_1, y_1)$, $Q = (x_2, y_2)$, and $x_1 = x_2$, $y_1 \neq y_2$, the line is vertical and intersects the curve at the point at infinity $\mathcal{O}$[28].
- **Addition with the Point at Infinity**: Adding any point P to $\mathcal{O}$ results in P[29]:

$$P + \mathcal{O} = P$$

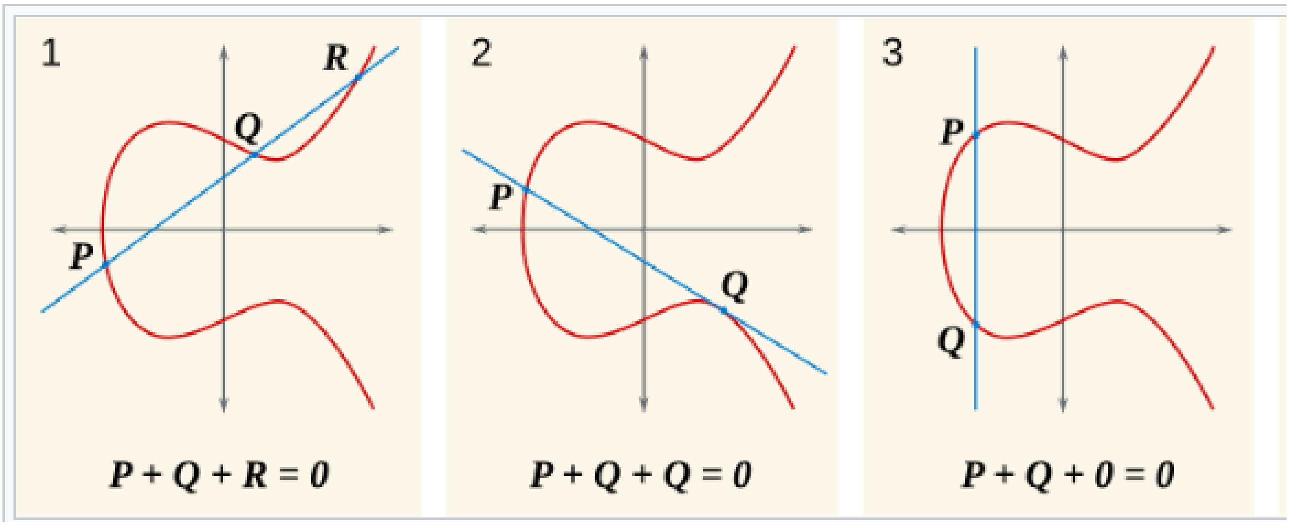


Figure 7: *Point Addition [30]*

Point Doubling

When doubling a point P, P is added to itself by drawing a tangent to the curve at P. Reflecting where the tangent crosses the curve on the x-axis doubles P[29].

Let $P = (x_1, y_1)$. The gradient m of the tangent line at P is given by:

$$m = \frac{3x_1^2 + a}{2y_1}$$

The equation of the tangent line at $P = (x_1, y_1)$ is:

$$y - y_1 = m(x - x_1)$$

The point $R = 2P = (x_3, y_3)$ lies on this line, and must also satisfy the elliptic curve equation. So, after substituting into (2), x_3 and y_3 are:

$$x_3 = m^2 - 2x_1$$

$$y_3 = m(x_1 - x_3) - y_1$$

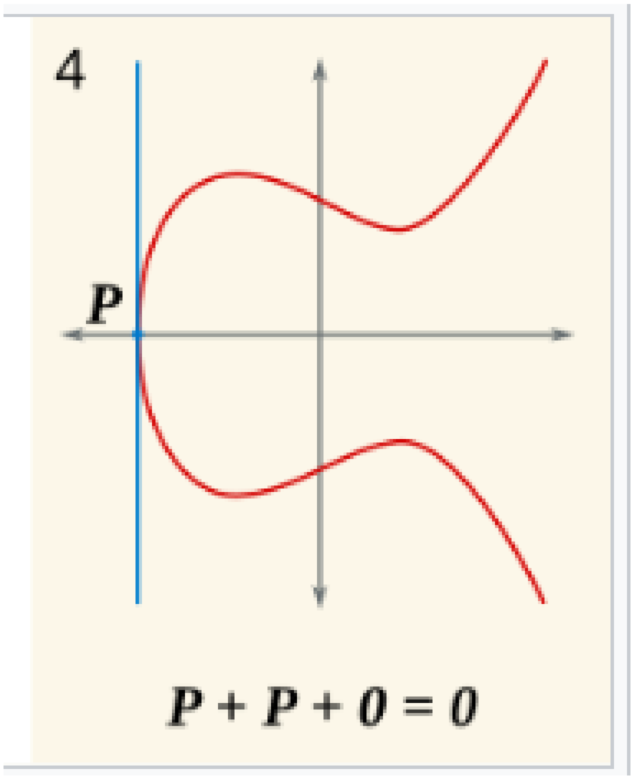


Figure 8: *Point Doubling [30]*

Point multiplication involves adding a point P to itself repeatedly. If k is an integer, the point multiplication kP is[31]:

$$kP = P + P + \ldots + P \text{ } (k \text{ times})$$

This uses the same process as point doubling.

Generator Point

A generator point G can generate several distinct points on the curve in point multiplication. If chosen correctly, any point on the curve can be computed through repeated addition of G[24].

$$G \in E(\mathbb{Z}/p\mathbb{Z})$$

where E is an elliptic curve modulo p that generates a cyclic subgroup, and $E(\mathbb{Z}/p\mathbb{Z})$ denotes the group of points on the elliptic curve.

A cyclic subgroup is a subset of points on the elliptic curve that can be expressed as multiples of the generator point G. For example, the cyclical subgroup contains:

$$\{\mathcal{O}, G, 2G, 3G, \ldots, (n-1)G\}$$

where n is the order of the generator point. This cyclic structure return to the identity element, after adding G to itself n-times. The order n of G is the smallest positive integer such that $nG = \mathcal{O}$[24].

The order, n, is important for security, determining how many distinct points can be generated by multiplying G by integers. A large prime order n ensures that the Elliptic Curve Discrete Logarithm Problem is hard to solve, ensuring security.

The cofactor, h, is the ratio of the total number of points on the elliptic curve N to the order of the generator point n, ensuring that cryptographic operations using the generator point G are efficient and secure.[24]:

$$h = \frac{N}{n}$$

The methods used to find the generator point are beyond the scope of this essay as they involve the Lagrange Theorem and Group Theory[24].

4.2.3 Elliptical Curve Discrete Logarithm Problem

Given an elliptic curve E defined over a finite field $\mathbb{F}_p$, a point P on the curve, and another point Q which is also on the curve, the ECDLP aims to find an integer k such that [24]:

$$Q = kP$$

The difficulty of solving the ECPLP arises from the one-way nature of scalar multiplication, and there are currently no efficient algorithms to solve it.

4.2.4 Elliptical Curve Diffie-Hellman protocol key exchange

Key Generation

Both parties agree on the following public information:

Steps	**Example**
Elliptic curve E over $\mathbb{F}_p$ with equation $y^2 = x^3 + ax + b$.	$e.g\ y^2 = x^3 + 2x + 3 \mod 17$
Generator point G on $E(\mathbb{F}_p)$.	$e.g\ G = (5, 1)$
Order n of the generator point G.	$e.g\ n = 19$

The generator point and order were chosen from known working examples, as calculating them is beyond the scope of the essay.

Each party selects a private key, which is a random integer in the range $[1, n-1]$.

Steps	**Example**
Alice's private key: a.	$a = 7$
Bob's private key: b.	$b = 11$

Encryption

Each party computes their public key by multiplying the base point G by its private key.

Steps	**Example**
Alice's public key: $A = aG$.	$e.g\ A = 7G = (13, 7)$
Bob's public key: $B = bG$.	$e.g\ B = 11G = (9, 16)$

Alice and Bob exchange their public keys.

Decryption

Each party computes the shared secret by multiplying the received public key by their private key.

Steps	**Example**
Alice computes: $S_A = aB$.	(Calculation performed, no explicit value given)
Bob computes: $S_B = bA$.	(Calculation performed, no explicit value given)

The shared secret is $S = S_A = S_B$, due to the associativity of the group.

$$e.g\ S = (15, 13)$$

Bob and Alice have the same point on the curve and can use this to encrypt messages.

5 Comparison of RSA encryption and ECC

5.1 Time Complexity and Key Sizes

5.1.1 Big O Notation

Big O notation is used to describe the algorithmic efficiency. It calculates time complexity, focusing on the worst-case scenario, providing an upper bound on the run-time[32][33].

An algorithm has time complexity $O(f(n))$ if there exist positive constants c and n_0 such that for all $n \geq n_0$:

$$T(n) \leq c \cdot f(n)$$

where:

- $T(n)$ is the actual run-time of the algorithm for an input of size n.
- $f(n)$ is a function describing how the run-time grows as n increases.
- c and n_0 are constants.

Common Big O Notations[5]

- $O(1)$: Constant time
- $O(\log n)$: Logarithmic time
- $O(n)$: Linear time
- $O(n \log n)$: Linearithmic time
- $O(n^2)$: Quadratic time
- $O(2^n)$: Exponential time

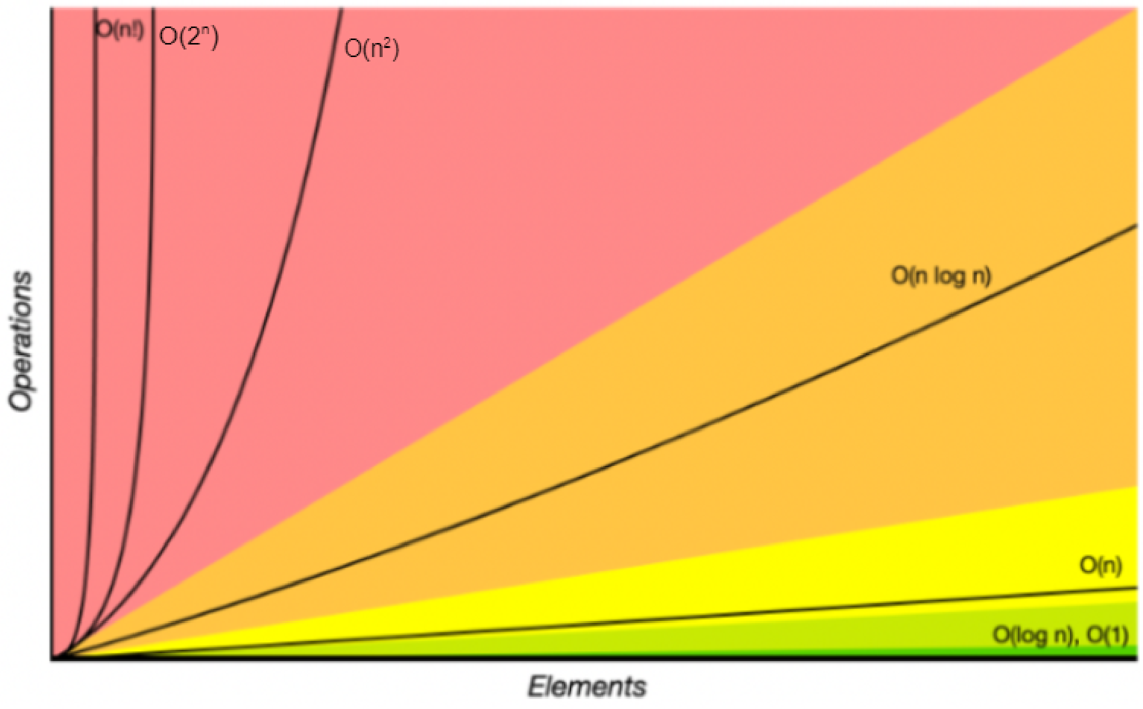


Figure 9: *Time Complexity Visual Comparisons [34]*

5.1.2 Time Complexity of RSA

Key Generation

- **Prime generation**: Finding large primes p and q is done using probabilistic primality tests like the Miller-Rabin test, which has a time complexity of $O(k \log^3 n)$ where k is the number of rounds.
- **Modulus computation**: Multiplying two n-bit numbers takes $O(n^2)$ using classical multiplication[35].

Encryption and Decryption

Both use modular exponentiation, where encryption performs $c = m^e \mod N$ and decryption performs $m = c^d \mod N$. Here, e and d are the public and private exponents, and N is the modulus.

Modular exponentiation is efficiently computed with exponentiation by squaring[34], as shown in **Appendix 1**. The steps are:

1. Write b in binary form: $b = (b_k b_{k-1} \dots b_1 b_0)_2$.
2. Initialise $result = 1$ and $base = a \mod n$.
3. For each bit b_i (left to right):
    - If $b_i = 1$, update $result = (result \cdot base) \mod n$.
    - Update $base = (base \cdot base) \mod n$.
4. The final value of $result$ is $a^b \mod n$.

The binary representation of b has $\log_2 b$ bits. Each iteration involves one multiplication and squaring operation, with time complexity $O((n)^2)$[36]. So, the total time complexity of modular exponentiation is $O(\log_2 b \cdot (n)^2)$.

For encryption, $b = e$ (public exponent), which is typically a small constant. Thus:

$$O((n)^2)$$

For decryption, $b = d$ (private exponent), can be as large as n. Thus:

$$O(\log_2 d \cdot (n)^2) = O((n)^3)$$

The time complexity for RSA encryption is $O((n)^2)$, and for decryption it's $O((n)^3)$.

5.1.3 Time Complexity of ECC

Key Generation

- **Private key**: A random integer k in the range $[1, n-1]$, so the time complexity is minimal.
- **Public key**: The point $Q = kG$ where G is the generator point on the curve, and the time complexity is $O(\log_2 k \cdot \log^2 p)$, as it uses point multiplication[6].

There are various algorithms for performing scalar multiplication, most commonly the Double-and-Add algorithm.

Double-and-Add Algorithm

The Double-and-Add algorithm steps are:

- Double the point P.
- Add the point P to itself conditionally based on the binary representation of k.

Each step involves either a point addition or doubling operation.

- Point Addition: Given two points P and Q, the time complexity is $O(\log_2 k)$[6].
- Point Doubling: Doubling a point leads to same time complexity $O(\log_2 k)$[6].

The number of steps is determined by the number of bits in k. If k has n bits, the number of steps is $O(n)$.
For each bit in k, either point doubling or addition is performed, so the total time complexity is:

$$O(n \cdot \log_2 k)$$

Since the number of bits in k is $n = \log_2 k$, the time complexity is:

$$O((\log_2 k)^2)$$

.

Comparison of RSA and ECC

Focusing on the time complexity solely, ECC time complexity is much lower, suggesting is significantly faster and efficient than RSA theoretically.

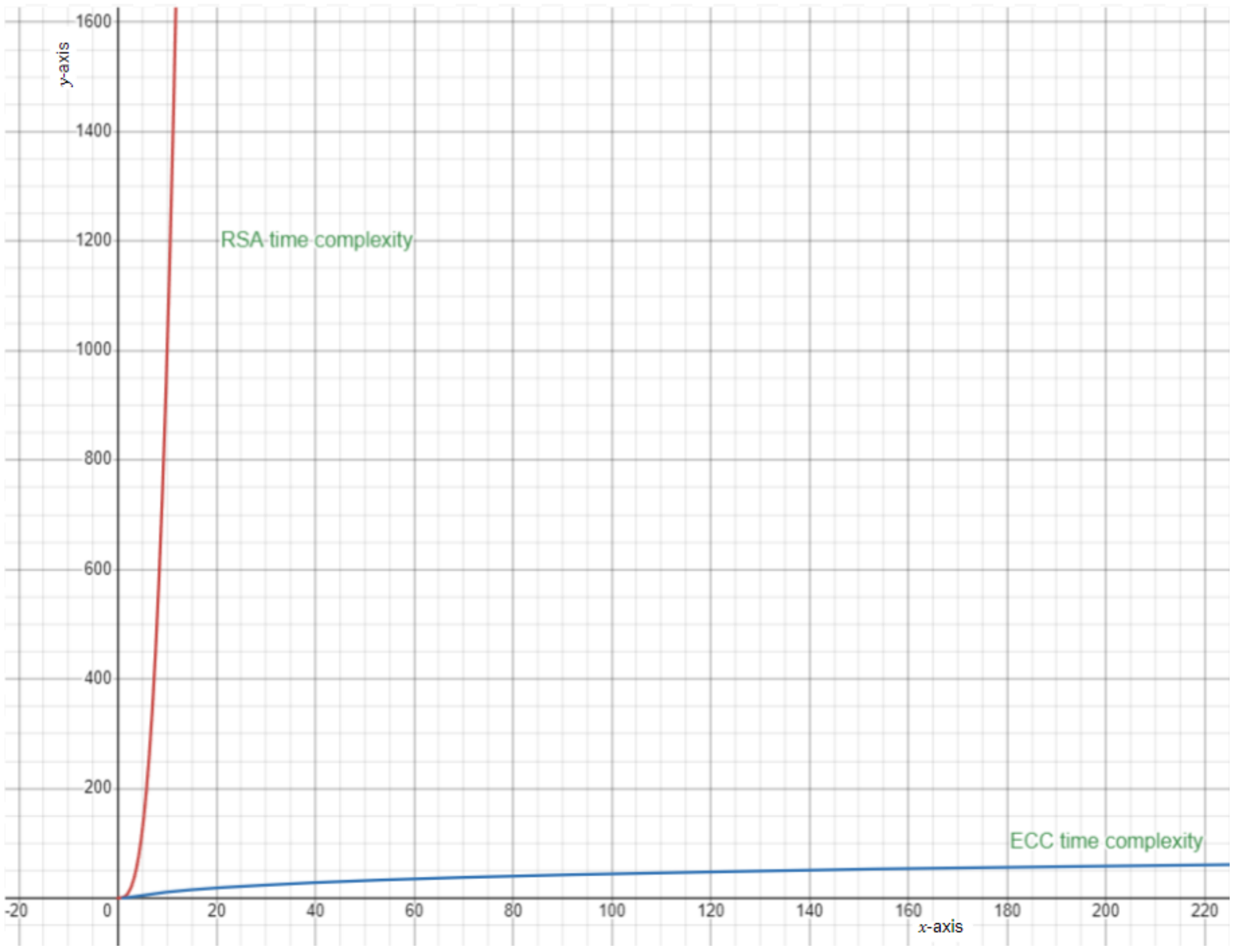


Figure 10: *RSA time complexity VS ECC time complexity*

Furthermore, other research papers have calculated and tested the time complexities for ECC and RSA[5].

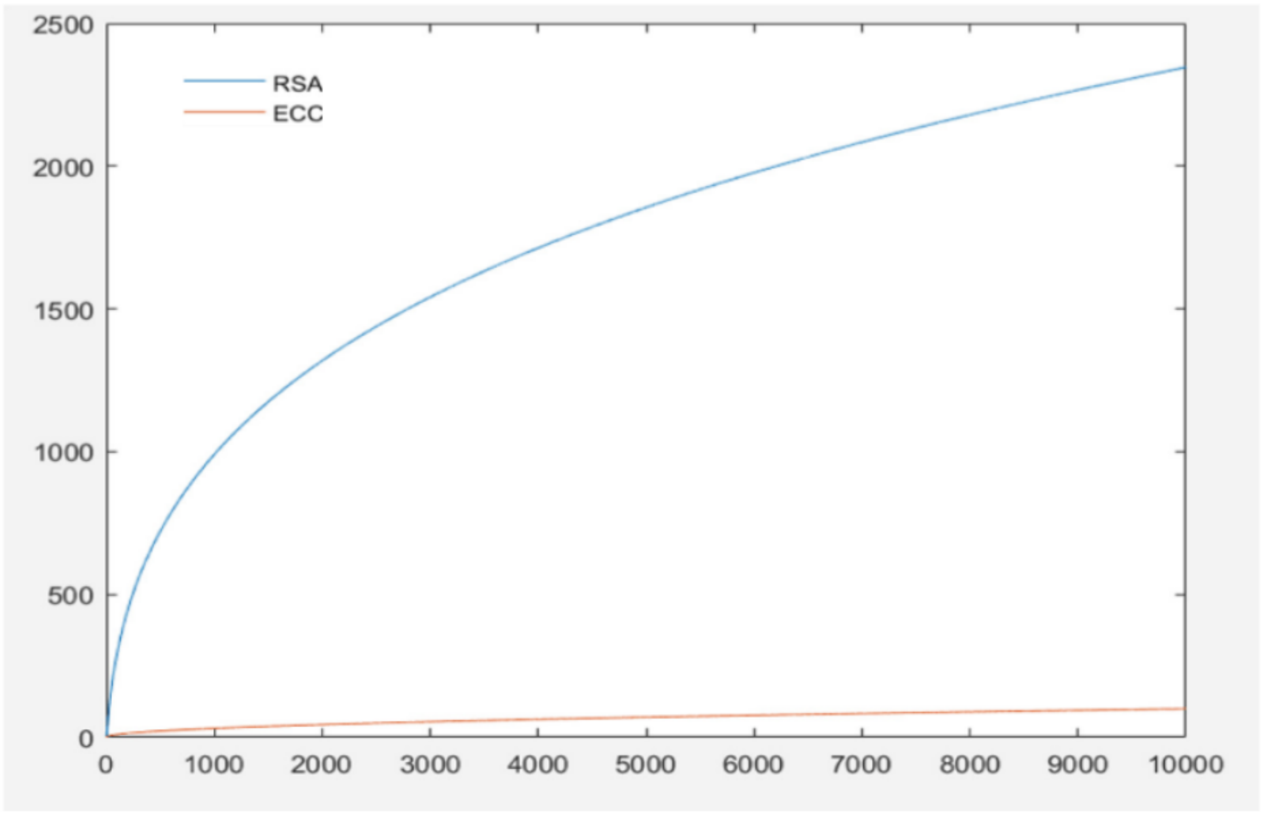
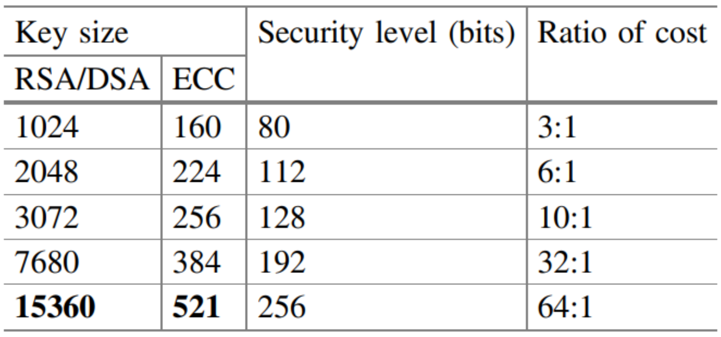


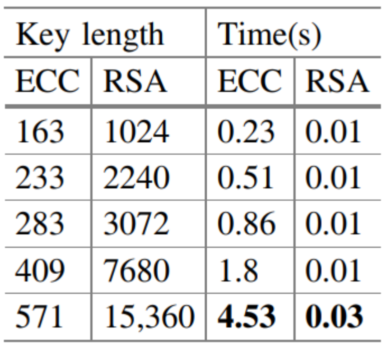
Figure 11: *RSA time complexity VS ECC time complexity[5]*

In addition, the key sizes for ECC are smaller than RSA. For example, to achieve a security level of 128 bits, RSA requires key sizes of around 3072 bits, but ECC only requires key sizes of 256 bits[5]. This affects the speed of the algorithm too, as shown in other research papers.



Key size		Security level (bits)	Ratio of cost
RSA/DSA	ECC		
1024	160	80	3:1
2048	224	112	6:1
3072	256	128	10:1
7680	384	192	32:1
15360	**521**	256	64:1

Figure 12: *RSA key size VS ECC key size[5]*



Key length		Time(s)	
ECC	RSA	ECC	RSA
163	1024	0.23	0.01
233	2240	0.51	0.01
283	3072	0.86	0.01
409	7680	1.8	0.01
571	15,360	**4.53**	**0.03**

Figure 13: *RSA key size VS ECC key size[5]*

Therefore, ECC is more efficient in terms key size for equivalent security levels and RSA requires significantly larger keys to achieve the same security, leading to slower performance.

5.2 Shor's Algorithm: Effect of Quantum computing on encryption

Shor's algorithm, developed by Peter Shor in 1994, is a quantum algorithm that efficiently factors large numbers. It uses quantum properties like superposition and interference to find periods, in functions, helping break down large composite numbers into their prime factors [37][38].

Quantum Terms and Concepts

- **Superposition**: In classical computing, a bit is either 0 or 1. In quantum computing, however, a qubit can be both 0 and 1 at the same time (superposition)[39]. Applying a Hadamard transform to qubits creates a superposition state, allowing them to represent multiple values at once[40].

- **Interference**: When qubits in superposition are measured, their wavefunctions collapse into specific states with probabilities influenced by their phase relationships[41]. This amplifies desired outcomes and suppress unwanted ones.

- **Quantum Fourier Transform (QFT)**: Similar to the classical Fourier transform, the QFT transforms a quantum state from the time domain (amplitudes) to the frequency domain (phase). It is crucial in extracting periodic information [42].

Shor's Algorithm consists of 2 parts[43]. The first is classical processing, reducing the problem to a period-finding problem. The second uses quantum mechanics to efficiently find the period[44].

5.2.1 Classical Preprocessing

Given an integer N, the goal is to find its prime factors. The steps are:

1. Choose a random integer a such that $1 < a < N - 1$.

2. Compute $\gcd(a, N)$. If $\gcd(a, N) \neq 1$, then $\gcd(a, N)$ is a non-trivial factor of N.

3. If $\gcd(a, N) = 1$, use quantum processing to find the period r of the function $f(x) = a^x \mod N$. The period r reveals the order of $a \mod N$, which be used with the Euclidean algorithm to factorize N.

5.2.2 Quantum Period Finding

Superposition:

Initialise two quantum registers. The first represents possible values of x and the second represents the function $f(x)$. Apply the Hadamard transform to the first register to create a superposition of all possible x values. This prepares the qubits to explore all potential inputs to the function $f(x)$ simultaneously.

Function Evaluation:

Apply a quantum operation to compute $f(x) = a^x \mod N$ and store the result in the second register. This uses quantum parallelism, allowing the quantum computer to evaluate $f(x)$ for all possible values of x simultaneously.

Interference and Measurement:

Measure the second register. This collapses the superposition to a state where the second register has a fixed value $f(x_0)$, leading to a superposition of states in the first register with the same $f(x)$. This sets up interference patterns in the first register, which will helps find the period of the function $f(x)$.

Quantum Fourier Transform:

Apply the QFT to the first register. The QFT maps the periodicity in the time domain to peaks in the frequency domain. It helps enhance the peaks corresponding to the period r:

Measurement and Period Extraction

Finally, measure the first register. The measurement will most likely collapse to a value close to a multiple of $\frac{Q}{r}$. Through several iterations, enough data is gathered to determine the period r using classical post-processing techniques.

5.2.3 Classical Postprocessing

Once the period r is found, check if r is even.
If r is odd, repeat the quantum period finding.
Compute $a^{r/2} \mod N$. If $a^{r/2} \equiv -1 \mod N$, repeat the quantum period finding.
Otherwise, compute the non-trivial factors of N as $\text{GCD}(a^{r/2} - 1, N)$ and $\text{GCD}(a^{r/2} + 1, N)$.

5.3 Impact on RSA and ECC

With Shor's Algorithm, the security of both encryption methods is severely at risk as they rely on the difficulty of factoring large numbers for their security. It solves both in polynomial time, and both are equally at risk to quantum computing in the future.

6 Conclusion

In conclusion, it is clear that RSA is relatively simple and quick to implement but is less efficient with a time complexity of $O((n)^2)$ and requires larger key sizes for a comparable level of security. ECC, on the other hand, is more complex to implement but offers greater efficiency with a time complexity of $O((\log_2 k)^2)$ and shorter key sizes for the same security level. Both RSA and ECC have their unique applications and benefits, making them suitable for different use cases. However, quantum computing poses a significant threat to both cryptographic systems. Shor's algorithm, with its polynomial-time efficiency in solving problems that are the foundation for RSA and ECC, highlights the need for future encryption methods to evolve and adapt.

Bibliography

1. Kessler, G. C. *An Overview of Cryptography* Accessed: 07/06/2024. `https://www.garykessler.net/library/crypto.html`.
2. Drummond, M. An Introduction to Cryptography. *IDPro Body of Knowledge* **13** (2024).
3. Bressoud, D. M. *The RSA Public Key Crypto-System* (Springer, New York, NY, 1989).
4. Hankerson, D. & Menezes, A. in *Encyclopedia of Cryptography, Security and Privacy* 1–2 (Springer, 2021).
5. Chandel, S. *et al. A Multi-dimensional Adversary Analysis of RSA and ECC in Blockchain Encryption* in *Advances in Information and Communication. FICC 2019* (eds Arai, K. & Bhatia, R.) **70** (Springer, Cham, 2020), 867–882.
6. Blake, I., Seroussi, G. & Smart, N. *Elliptic Curves and Their Applications to Cryptography: An Introduction* (Springer, 1999).
7. Rosen, K. H. *Elementary number theory* (Pearson Education London, 2011).
8. Rivest, R. L., Shamir, A. & Adleman, L. A Method for obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM* **21,** 120–126 (1978).
9. Javatpoint. *RSA Encryption in Discrete Mathematics* Accessed: 04/06/2024. `https://www.javatpoint.com/rsa-encryption-in-discrete-mathematics`.
10. Burton, D. M. *Elementary Number Theory* 7th (McGraw-Hill Higher Education, New York, 2011).
11. Williamson, J. *The Elements of Euclid: With Dissertations* (UMI, 2003).
12. Ore, O. *Number theory and its history* (Courier Corporation, 2012).
13. Heath, T. L. *The Thirteen Books of Euclid's Elements* (Dover Publications, Inc, 1956).
14. Pettofrezzo, A. J. & Byrkit, D. R. *Elements of Number Theory* (Prentice-Hall, 1970).
15. Stark, H. M. *An introduction to number theory* (MIT Press, 1978).

16. Long, C. T. *Elementary Introduction to Number Theory* 2nd (D. C. Heath, 1972).

17. Smart, N. P. The discrete logarithm problem on elliptic curves of trace one. *Journal of cryptology* **12,** 193–196 (1999).

18. Hardy, G. H. & Wright, E. M. *An introduction to the theory of numbers* (Oxford university press, 1979).

19. Boneh, D. *et al.* Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS* **46,** 203–213 (1999).

20. Menezes, A. J., Van Oorschot, P. C. & Vanstone, S. A. *Handbook of applied cryptography* (CRC press, 2018).

21. Stallings, W. *Network and internetwork security: principles and practice* (Prentice-Hall, Inc., 1995).

22. Vagle, J. L. A Gentle Introduction to Elliptic Curve Cryptography. *Cambridge (MA): BBN Technologies* (Nov. 2000).

23. Fraleigh, J. *A First Course in Abstract Algebra* (Pearson Education, 2003).

24. Nakov, S. *Elliptic Curve Cryptography (ECC)* Accessed: 04/07/2024. `https://cryptobook.nakov.com/asymmetric-key-ciphers/elliptic-curve-cryptography-ecc#elliptic-curves-over-finite-fields`.

25. Blake, I. F., Seroussi, G. & Smart, N. P. *Advances in elliptic curve cryptography* (Cambridge University Press, 2005).

26. Davis, T. R. *Elliptic Curve Cryptography* Mathematical Circles Topics, 3 Nov. 2013, `http://www.geometer.org/mathcircles/ecc.pdf`. 2013.

27. Silverman, J. H. *The arithmetic of elliptic curves* (Springer, 2009).

28. Hankerson, D., Vanstone, S. & Menezes, A. Cryptographic protocols. *Guide to Elliptic Curve Cryptography,* 153–204 (2004).

29. Blake, I. *Elliptic Curves in Cryptography* (Cambridge University Press, 1999).

30. Of Bits, T. *Bluetooth Invalid Curve Points* Accessed: 12/07/2024. `https://blog.trailofbits.com/2018/08/01/bluetooth-invalid-curve-points/`.

31. Washington, L. C. *Elliptic curves: number theory and cryptography* (Chapman and Hall/CRC, 2008).

32. Mohr, A. Quantum Computing in Complexity Theory and Theory of Computation. *Carbondale, IL* **1** (2014).

33. Huang, S. *Big O Notation: Why it Matters and Why it Doesn't* Accessed: 24/05/2024. `https://www.freecodecamp.org/news/big-o-notation-why-it-matters-and-why-it-doesnt-1674cfa8a23c/`.

34. GeeksforGeeks. *Multiplicative Inverse Under Modulo m* Accessed: 18/06/2024. `https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/`.

35. GeeksforGeeks. *Fast Modular Exponentiation* Accessed: 19/06/2024. `https://www.geeksforgeeks.org/fast-exponentiation-iterative-method/`.

36. Knuth, D. E. *The Art of Computer Programming: Seminumerical Algorithms, Volume 2* (Addison-Wesley Professional, 2014).

37. Shor, P. W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* **41,** 303–332 (1999).

38. Shor, P. W. *Algorithms for Quantum Computation: Discrete Logarithms and Factoring* in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994), 124–134.

39. Yanofsky, N. S. & Mannucci, M. A. *Quantum computing for computer scientists* (Cambridge University Press, 2008).

40. Nielsen, M. A. & Chuang, I. L. *Quantum computation and quantum information* (Cambridge university press Cambridge, 2001).

41. Lipton, R. J. & Regan, K. W. Quantum algorithms via linear algebra. *MIT Press eBook* **47,** 2993749–2993752 (2014).

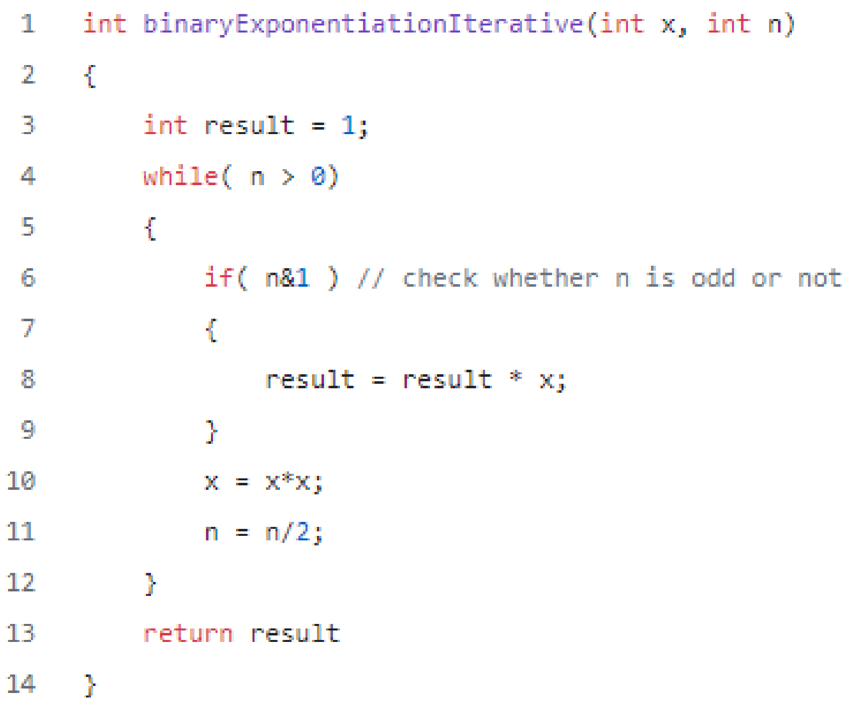
42. Hidary, J. D. & Hidary, J. D. *Quantum computing: an applied approach* (Springer, 2019).

43. Classiq. *Shor's Algorithm Explained* Accessed: 23/05/2024. `https://www.classiq.io/insights/shors-algorithm-explained`.

44. GeeksforGeeks. *Shor's Factorization Algorithm* Accessed: 30/09/2024. `https://www.geeksforgeeks.org/shors-factorization-algorithm/`.

45. Kumar, A. *Modular Exponentiation* Accessed: 08/08/2024. `https://medium.com/@anilkumar78/modular-exponentiation-8b5bd7b16042`.

1 Appendix



```
int binaryExponentiationIterative(int x, int n)
{
    int result = 1;
    while( n > 0)
    {
        if( n&1 ) // check whether n is odd or not
        {
            result = result * x;
        }
        x = x*x;
        n = n/2;
    }
    return result
}
```

Figure 14: *Illustration of the modular exponentiation algorithm [45]*