Birla Vishvakarma Mahavidyalaya Engineering College

(An Autonomous Institution)

Course: Computer Networks(2CP07)

RSA ALGORITHM IN CRYPTOGRAPHY

Prepared By:

Rajveer Rathod(19CP074)

Akshat Trivedi(19CP075)

Vatsal Dhupelia(19CP076)

Submitted To:

Prashant B Swadas

Department of Computer Engineering

Semester:4 Batch: D

Academic Year: 2020-21

Abstract

Cryptography plays a huge role in our highly technological daily life, and we are profoundly depending on the science of hiding information in plain sight. There are numerous ways to achieve this, where number theory plays a huge role in cryptography to ensure that information cannot be easily recovered without special knowledge.

One of the most reliable and secure encryption algorithms available today is the RSA algorithm, which provides great encryption and performance using asymmetric cryptography, also known as public-key cryptography.

This report focuses on the mathematics behind the algorithm, along with its core functionality and implementation. In addition, the code implementation and the encryption and decryption procedure are provided in details.

Table of Contents

1 Introduction

Achieving the goal of encrypting messages to hide information in plain sight can be done in many ways. Cryptography has existed for thousands of years and the evolution and science of cryptography has improved exceedingly over the past few decades. There are many algorithms available that can be used to encrypt and decrypt messages, where one of the most used algorithms is called RSA.

The algorithm was introduced by three researchers in 1976, named Ronald Rivest, Adi Shamir and Leonard Adleman, and is based on encrypting messages using modular exponentiation, and the sharing of public and private keys. In a symmetric encryption algorithm, there is a secret key that is used to both encrypt and decrypt the data. Symmetric encryption is relatively fast, but this method of encryption can cause problems in terms of distributing the secret key, since it needs to be used for both encrypting and decrypting the data.

Unlike symmetric algorithms, such as for example AES, public key algorithms require the computation of both the public and private keys. What makes RSA so special compared to other encryption algorithms is that these keys must be computed using mathematics, and are not random numbers that are generated.

Along with symmetric cryptography, there is another method called asymmetric cryptography, where there is a private key and a public key used during the encryption process. The public key is obviously public and does not need to remain secret, while the private key must be kept in between the involved parties of the message transmission. So, anyone can encrypt a message using the public key, but only the ones who know the private key can decrypt the message. Otherwise, it will remain encrypted. This method solves some key-sharing challenges and makes it easier to share the keys.

The key generation part of the RSA algorithm is quite central and important, and this is something that's missing in most symmetric key algorithms, where the key generation part is not really complicated in terms of mathematical computations.

RSA is today used in a range of web browsers, chats and email services, VPNs and other communication channels. It is commonly used simply because people trust the algorithm to provide good enough encryption for their purposes, and it has been proven to be secure.

2 Cryptography

Cryptography is the science of keeping information secure by transforming it into form that unintended recipients cannot understand. In cryptography, an original human readable message, referred to as plaintext, is changed by means of an algorithm, or series of mathematical operations, into something that to an uninformed observer would look like gibberish; this gibberish is called ciphertext.

Cryptographic systems require some method for the intended recipient to be able to make use of the encrypted message — usually, though not always, by transforming the ciphertext back into plaintext.

2.1 What is Cryptography

This is all very abstract, and a good way to understand the specifics of what we're talking about is to look at one of the earliest known forms of cryptography. It's known as the Caesar cipher, because Julius Caesar used it for his confidential correspondence; as his biographer Suetonius described it, "if he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet ... If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others."

Suetonius's description can be broken down into the two cryptographic elements we've discussed, the algorithm and the key. The algorithm here is simple: each letter is replaced by another letter from later in the alphabet. The key is how many letters later in the alphabet you need to go to create your ciphertext. It's three in the version of the cipher Suetonius describes, but obviously other variations are possible — with a key of four, A would become E, for instance.

A few things should be clear from this example. Encryption like this offers a fairly simple way to secretly send any message you like. Contrast that with a system of code phrases where, say, "Let's order pizza" means "I'm going to invade Gaul." To translate that sort of code, people at both ends of the communication chain would need a book of code phrases, and you'd have no way to encode new phrases you hadn't thought of in advance. With the Caesar cipher, you can encrypt any message you can think of. The tricky part is that everyone communicating needs to know the algorithm and the key in advance, though it's much easier to safely pass on and keep that information than it would be with a complex code book.

The Caesar cipher is what's known as a substitution cipher, because each letter is substituted with another one; other variations on this, then, would substitute letter blocks or whole words. For most of history, cryptography consisted of various substitution ciphers deployed to keep government and military communications secure. Medieval Arab mathematicians pushed the science forward, particularly the art of decryption — once researchers realized that certain letters in a given language are more common than others, it becomes easier to recognize patterns, for instance. But most pre-modern encryption is incredibly simple by modern standards, for the obvious reason that, before the advent of computers, it was difficult to perform mathematical transformations quickly enough to make encryption or decryption worthwhile.

In fact, the development of computers and advances in cryptography went hand in hand. Charles Babbage, whose idea for the Difference Engine presaged modern computers, was also interested in cryptography. During World War II, the Germans used the electromechanical Enigma machine to encrypt messages — and, famously, Alan Turing led a team in Britain that developed a similar machine to break the code, in the process laying some of the groundwork for the first modern

computers. Cryptography got radically more complex as computers became available, but remained the province of spies and generals for several more decades. However, that began to change in the 1960s.

2.2 Cryptography in Network Security

It was the formation of the first computer networks that started civilians thinking about the importance of cryptography. Computers were talking to each other over the open network, not just via direct connections to one another; that sort of networking was transformative in many great ways, but also made it trivially easy to snoop on data traveling across the network. And with financial services being an early use case for computer communication, it was necessary to find a way to keep information secret.

IBM led the way in the late 1960s with an encryption method known as "Lucifer", which was eventually codified by the US National Bureau of Standards as the first Data Encryption Standard (DES). As the internet began to grow in importance, more and better encryption was needed, and today a significant portion of data flying around the world is encrypted using varying techniques.

2.3 What is cryptography used for?

We've already discussed some of the specific applications of cryptography, from keeping military secrets to transmitting financial data safely across the internet. In the bigger picture, though, there are some broad cybersecurity goals that we use cryptography to help us achieve, as cybersecurity consultant Gary Kessler explains. Using cryptographic techniques, security pros can:

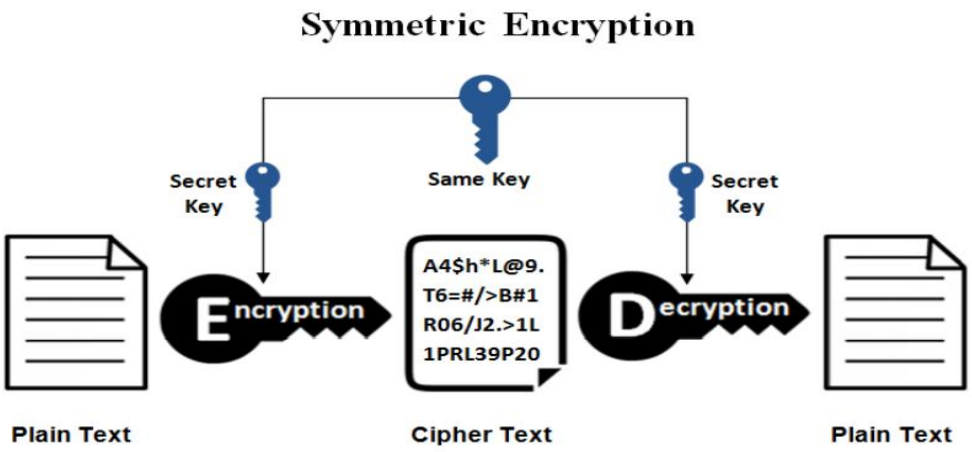
- Keep the contents of data confidential
- Authenticate the identity of a message's sender and receiver
- Ensure the integrity of the data, showing that it hasn't been altered
- Demonstrate that the supposed sender really sent this message, a principle known as non-repudiation

You may recognize some of these principles from variations of the CIA triad. The first of these uses is the obvious one — you can keep data secret by encrypting it. The others take a bit of explanation, which we'll get into as we describe the different types of cryptography.

2.4 Types of Cryptography:

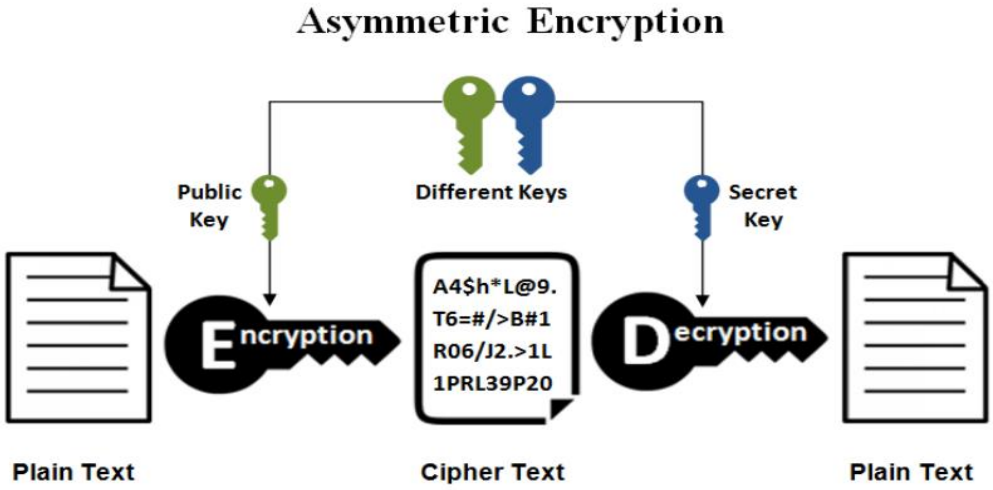
2.4.1 Symmetric and Asymmetric Key Cryptography

In a symmetric encryption algorithm, there is a secret key that is used to both encrypt and decrypt the data. Symmetric encryption is relatively fast, but this method of encryption can cause problems in terms of distributing the secret key, since it needs to be used for both encrypting and decrypting the data. If Alice sends a symmetric-encrypted message to Bob, she needs to inform him about the secret key as well, so he can use it to decrypt the message. This might lead to some unnecessary challenges. There are many existing symmetric encryption algorithms, such as Caesar cipher, AES and DES. Some of these algorithms are still used today and can be relied upon, as symmetric encryption is safe and fast enough for most purposes.



Along with symmetric cryptography, there is another method called asymmetric cryptography, where there is a private key and a public key used during the encryption process. The public key is obviously public and does not need to remain secret, while the private key must be kept in between the involved parties of the message transmission. So, anyone can encrypt a message using the public key, but only the ones who know the private key can decrypt the message. Otherwise, it will remain encrypted. This method solves some key-sharing challenges and makes it easier to share the keys. Examples of this algorithm are RSA, ElGamal encryption and Diffie-Hellman algorithm. If we compare symmetric and asymmetric encryption, we can see that asymmetric is a bit slower due to the fact that it handles two keys and it takes some more time to process this.

It is important to keep in mind that both symmetric and asymmetric encryption are secure and cannot compete or be compared directly, because they both serve a great purpose for different use cases. Some individuals might prefer symmetric because it is simple and provides enough security for their purpose, while other prefer asymmetric due to its key distribution method. Both types of encryption have pros and cons, where for example symmetric encryption is faster than asymmetric, while it is weak in terms of distributing the keys to the involved parties.



However asymmetric encryption settles the key distribution challenge, but it has the disadvantage of being slow by comparison to symmetric encryption. After all, both encryption methods are used in today's modern technology and they both serve a great purpose in terms of confidentiality and integrity.

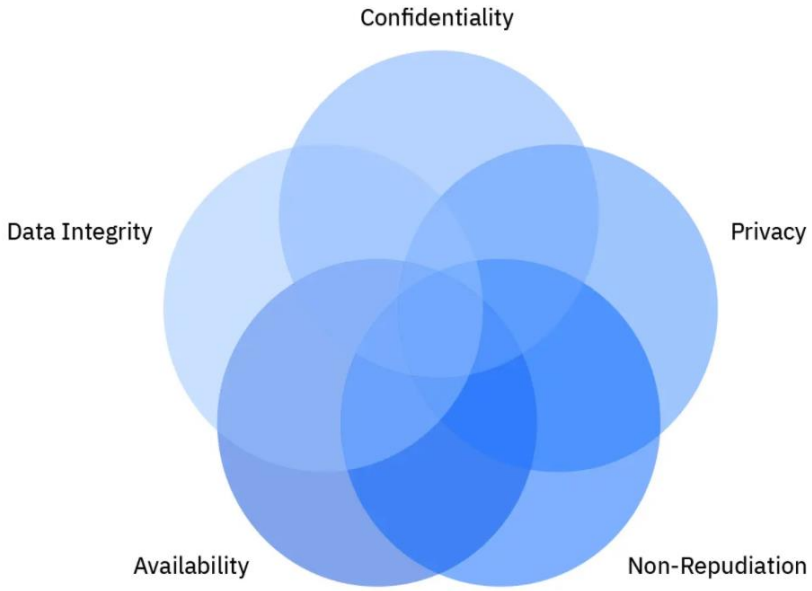
A practical example of asymmetric cryptography:

1. A client such as a web browser submits its public key to the server, and makes a request for some data.
2. The server responds to the web browser by encrypting the data using the same public key.
3. The server sends this encrypted data back to the client.
4. The client receives this encrypted data and decrypts it using the private key.

Since this process is asymmetric, no one else except the client (web browser) can decrypt the data, even if a third-party individual has access to the public key. This same concept will be introduced in the RSA algorithm as well.

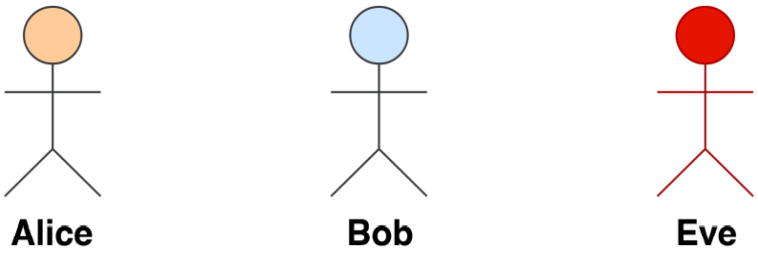
2.5 A Practical Example

The CIA triad is a security model that stands for Confidentiality, Integrity and Availability, and is the primary focus in information security to balance the protection of online information. The CIA triad is very fundamental in cybersecurity and the three concepts should be guaranteed in any secure system in order to avoid sensitive data breaches and loss of personal information in today's modern technology.



We will demonstrate the concepts of CIA through a practical example using two actors: Alice and Bob. There is actually a third actor as well, the eavesdropper called Eve. I will use these three names during the illustration of this practical example. Alice wants to send the message" USN Kongsberg is best!" to Bob. They are both using a safe messaging app on their phones, and from the moment Alice submits the message to the moment Bob receives it, there are some steps that

process in the background in terms of confidentiality, integrity and availability in order to make sure that the message gets transferred to Bob without any security breaches or failures.



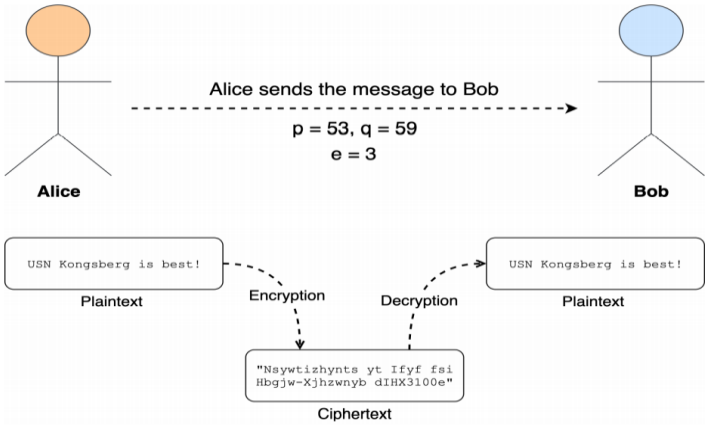
The goal here is to make sure that the message Alice submits is safely sent to Bob, without anyone else interfering with the message transmission. This is done by cryptography, where the communication is encrypted and protected, so that only Alice and Bob can understand the message that is being sent. So even if for example Eve does manage to interfere the message transmission, it is encrypted and not readable as plain text.

There are numerous different ways in cryptography to perform an encryption, both secure and weak methods. We usually talk about two algorithms in cryptography: symmetric and asymmetric algorithms. An algorithm is basically a formula or a procedure to solve a specific problem, which in this case is encryption on data.

In symmetric algorithms it is required that both the sender and the receiver, Alice and Bob, must have the same key and the same processing algorithm as well. This symmetric key is often called a secret key or a private key, which is kept protected from anyone else except the sender and the receiver of the encrypted message. If the private key is revealed to Eve, the secure encryption is compromised and the message can be decrypted by anyone whom may have access to the key.

However, in asymmetric algorithms there are two keys: a public key and a private key. Alice uses the public key to encrypt the message and Bob uses the private key to decrypt it. It is not dangerous if the public key is truly public, but the private key however must be kept private and only the owner can know what the private key is. If both public and private keys are announced to anyone else except Alice and Bob, the encrypted message will no longer be secure and it can be decrypted at any time.

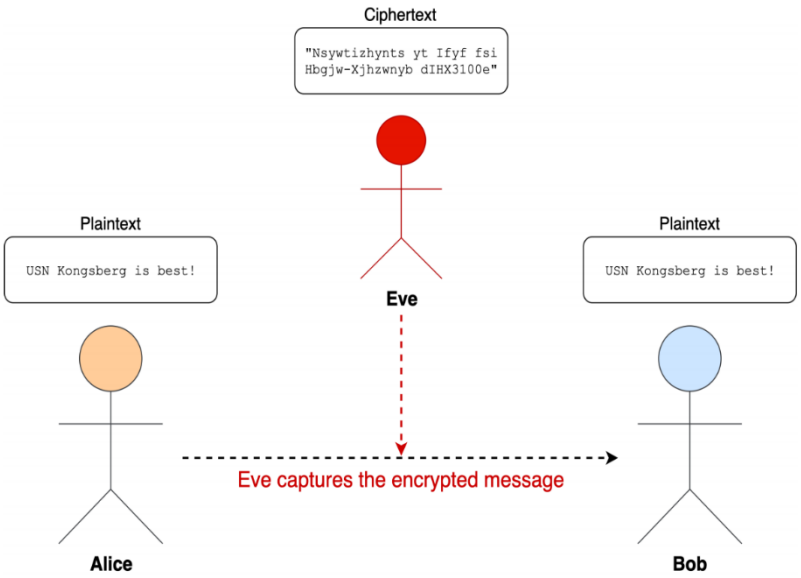
In this scenario I will use the RSA algorithm to demonstrate how the message is being encrypted and decrypted between Alice and Bob. In order to fulfil the first term in the CIA triad, confidentiality, we need to encrypt the message Alice sends to Bob in order to make sure that the message is hidden from anyone whom is not authorized to read it. Alice writes the message in plaintext and sends it to Bob. The messaging app prepares the message by encrypting it using RSA.



As soon as Bob receives the message, the mobile app decrypts the ciphertext using the same algorithm that Alice used to encrypt the message. The receiver knows the public key values, so it decrypts the ciphertext to plaintext, and shows the results to Bob as soon as he reads the message on his phone. And as expected, the message Bob reads is" USN Kongsberg is best!".

Okay, so what if there is another actor involved in this scenario? There is Eve, a mysterious woman, who is secretly monitoring Alice's network activities. Eve is sniffing packets across the computer network and has successfully captured the network packets containing the private message Alice sent to Bob. This is where we come back to the CIA triad and the Data confidentiality term, where information is hidden in terms of encryption to avoid any unauthorized third party from reading it, such as Eve in this scenario.

Even though Eve has captured the message Alice sent to Bob, it is written in ciphertext and therefore not decryptable unless Eve knows the private key that was used during the message transfer.



3 Introduction to Number Theory

We will introduce some of the number theory and cryptography concepts used in the RSA algorithm, as a brief mathematical introduction to the algorithm and its core functionality.

3.1 GCD and Prime Numbers

Prime numbers have a huge role in the RSA algorithm, and numbers such as 2, 3, 5, 6, 11, 13, 17, ..., are natural numbers greater than 1 that cannot be expressed as a product of other smaller natural numbers. Per definition, a prime is an integer greater than 1 that is divisible by no positive integers other than 1 and itself. Prime numbers go back to ancient times, where it was known already thousands of years ago that there are infinitely many existing primes.

Primes are today very essential in modern cryptographic systems, and consist many important properties in cryptography. It is important to keep in mind that the number 1 is not a prime number. Prime numbers are specifically used in the key generation process of the RSA algorithm, and really is what the entire algorithm is based upon.

The Greatest Common Divisor (GCD) of two or more integers is the greatest positive integer that divides each of the integers. For example, the GCD of 53 and 59 is 1. If *gcd(a, b) = 1*, we say that a is coprime to b. For example *gcd(3, 59) = 1*, which indicates that 3 is coprime to 59, which also means that 59 is coprime to 3.

For large numbers, the greatest common divisor can quickly be calculated by using the Euclidean algorithm. The algorithm shows that *d = gcd(a, b)* always can be written as *d = sa + tb*, where *s* and *t* are positive integers. Computing the greatest common divisor of two integers by directly using the prime factorization of these integers are proven to be quite inefficient. This way of calculating prime numbers is time-consuming and therefore the Euclidean algorithm is often used for large numbers, since it provides a more elegant way of computing the greatest common divisor.

If we are able to show that the common divisors of a and b are the same as the common divisors of b and r, it has been proven that *gcd(a, b) = gcd(b, r),* due to the fact that both of these pairs must consist of the same greatest common divisors. To show an example for how the greatest common divisor of two integers can be calculated using the Euclidean algorithm:

$$662 = 414 \cdot 1 + 248$$

$$414 = 248 \cdot 1 + 166$$

$$248 = 166 \cdot 1 + 82$$

$$166 = 82 \cdot 2 + 2$$

$$82 = 2 \cdot 41$$

The calculations prove that the greatest common divisor of *(414, 662) = 2*, because *2* is the last remainder.

3.2 Fermat's Little Theorem

One of the basic theorems of number theory used in the RSA algorithm is Fermat's little theorem. Pierre de Fermat was one of the greatest contributors of mathematics and is known for his number theories. He contributed with one very famous theorem in number theory, known as Fermat's last theorem.

This theorem states that, for any integer $n \geq 3$, the equation $x^n + y^n = z^n$ has no solution with x, y and z all positive integers. Fermat's little theorem is very central in the RSA algorithm, and states that a $p \equiv a \pmod p$ for every integer a and for every prime p.

If p is a prime number, then $2^p - 1 \equiv 1 \pmod p$. For example, to prove that the number 35 is not a prime, it can be shown by successive squaring.

$$2^4 \equiv 16$$

$$2^8 \equiv 256 \equiv 11$$

$$2^1\,6 \equiv 121 \equiv 16$$

$$2^3\,2 \equiv 256 \equiv 11$$

Therefore, it can be concluded that $2^3\,4 \equiv 2^{32}.2^2 \equiv 11.4 \equiv 9 \neq 1 \pmod{35}$. In other words, it is not a prime number. This is a quick and effective method of proving that 35 is composite without necessarily finding its factors. Fermat's theorem will be mentioned multiple times throughout the explanation of the RSA algorithm, as it contributes with many important properties in modern cryptography.

3.3 Modular Arithmetic

Often in number theory we only care about the remainder of an integer when the integer is divided by another positive integer. Since we often only care about the remainders, there is a special notation for them as well. The notation a mod m is used to represent the remainder when an integer a is divided by another positive integer m.

Another related notation is often used, that indicates that two integers have the same remainder when the integers are divided by another positive integer m. This indicates that if a and b are both integers, and m is a positive integer, then $a \equiv b \pmod m$ if and only if $a \bmod m = b \bmod m$.

These modular arithmetic equations will be used repeatedly in the RSA algorithm, as the encryption and decryption methods depend on the modulo operator.

3.4 Euler's Theorem and Totient Function

Earlier the Fermat's little theorem was introduced where the theorem says that if p is a prime number and a is not a multiple of p, then $a^{p-1} = 1 \pmod p$. However, Euler's generalization of

Fermat's theorem indicates that if a is relatively prime to m, then $a^{\Phi(m)} = 1 \pmod{m}$, where $\varphi(m)$ is called a totient function by Euler.

This so-called totient function will count the number of positive integers relatively prime and less than m. Euler's theorem is used in the RSA encryption process, where two enormous prime numbers p, q are used, whereas p and q are kept private, and $n = p \cdot q$ is kept public. The private key d however solves for a public key e, such that $de = 1 \pmod{\varphi(n)}$. Since the p and q values are known, it is possible to compute $\varphi(n) = (p - 1) \cdot (q - 1)$, along with computing the public key e.

Euler's theorem comes in handy once again when someone wants to send a message m, using the public key e, such that $m^e \pmod{n}$. Accordingly, the computation of the encrypted message happens by Euler's theorem as well, such that $(m^e)^d = m^{\varphi(n)} = m \pmod{n}$.

There are many use cases for Euler's theorem and totient function in number theory, as it is commonly used in primality testing too, where it checks and proves if a number is not a prime. The totient function, or Phi function, often occurs in practical applications, and is very much used in modern cryptography.

3.5 Extended Euclidean Algorithm

The Euclidean algorithm was mentioned earlier, where it was used to calculate the greatest common divisors, and now there is an extended Euclidean algorithm, which essentially is the Euclidean algorithm ran backwards. As an extension to the Euclidean algorithm, it computes the coefficients of Bezout's identity, which are integers x and y such that $ax + by = gcd(a, b)$. This algorithm is especially useful when integers a and b are coprime, and the extended Euclidean algorithm is widely used in modern cryptography, specifically the RSA algorithm where it computes the modular multiplicative inverse in the public-key encryption and decryption methods.

It is possible to find the integers x and y by reversing the steps in the Euclidean algorithm. The initial idea is to start with the greatest common divisor and recursively work itself backwards, where it eventually finds a result for x and y, which are two integers.

4 The RSA algorithm

RSA (Rivest-Shamir-Adleman) is an asymmetric cryptographic algorithm used to encrypt and decrypt messages by modern computers. Asymmetric states that there are two different keys used in the encryption and decryption process, which also is called public-key cryptography. This is simply because one of the two keys can be given to anyone without exploiting the security of the algorithm.

The RSA algorithm involves both private and public keys. The public key can be known and published to anyone, as it is used to encrypt the messages from plaintext to ciphertext. The messages that are encrypted with this specific public key can however only be decrypted with the corresponding private key. The key generation process of the RSA algorithm is what makes it so secure and reliable today, as it contains a high level of complexity compared to other cryptographic algorithms.

4.1 Key generation

Unlike symmetric algorithms, such as for example AES, public key algorithms require the computation of the pair (K_{public}, $K_{private}$). What makes RSA so special compared to other encryption algorithms is that these keys must be computed using mathematics, and are not random numbers that are generated.

The key generation part of the RSA algorithm is quite central and important, and this is something that's missing in most symmetric key algorithms, where the key generation part is not really complicated in terms of mathematical computations.

The key generation process of the RSA algorithm consists of five steps:

1) Choose two large prime numbers (p and q)
2) Compute $n = p \cdot q$
3) Calculate $\varphi(n) = (p - 1) \cdot (q - 1)$
4) Choose an integer e such that $1 < e < \varphi(n)$, and:
   a) Ensure that $gcd(e, \varphi(n)) = 1$
   b) Ensure that e and $\varphi(n)$ are coprime
5) Compute an integer d, such that $d = e^{-1} \bmod \varphi(n)$

After completing these five steps, we have generated two asymmetric keys that can be used for encryption and decryption further in the process: the public key consists of n and e, while the private key consists of d.

$$K_{public} = (n, e) \qquad K_{private} = (d)$$

It is common practice to use large numbers in the generation process for p and q, often at least 512 bits, which makes the n 1024 bits ($n = p \cdot q$). This increases the level of complexity and makes the cryptanalysis process considerable harder in terms of Bruce force attacks.

4.2 Encryption and decryption

After computing all the necessary variables for the key generation, it is time to encrypt and decrypt a message using the algorithm. This is of course given the fact that K_{public} has been generated, and consists of n and e. The formula is very simple for both the encryption and decryption process, which states that:

Encryption: $c \equiv m^e \pmod n$, where m is the message in plaintext.

Decryption: $m \equiv c^d \ (mod\ n)$, where c is the ciphertext.

Anyone who wants to encrypt a message needs to use the public key of the recipient, in order to ensure that the message is only decryptable by the correct individual so that it only decrypts with a specific private key. The recipient shares the public key, while keeping the private key secret.

The sender then wants to submit a message M, which gets transformed into a smaller number than n, and this is done by a reversible protocol known as a padding scheme. The message gets computed into an encrypted ciphertext, which at last gets submitted over to the recipient.

The padding scheme used in the encryption process is quite important, and without this scheme there would be some problems. The padding scheme ensures that no values of the message are insecure, such as for example the values $m = 0$ or $m = 1$ will respectively compute ciphertexts equal to 0 or 1, which is caused by the properties of the exponentiation. When the exponent used in the key generation process is small, this might cause the non-modular result of m^e to be less than the modulus n. This means that ciphertexts may be brute forced and decrypted easily by calculating the e^{th} root of the ciphertext without necessarily regarding the modulus.

For example, when encrypting a text with the numeric value of 0, it would encode as $m = 0$, which then again computes the ciphertext $c = 0$, with no concern about the values of n and e. The same goes for the numeric value of 1, which produces the value of 1 in ciphertext. This creates an insecure pattern, which might be analysed by attackers and easily decrypted after gaining some knowledge about the encryption process.

To avoid such problems in the algorithm, it is common to implement a randomized padding into the message before the encryption happens. This is to ensure that the message does not contain some insecure values and that the encrypted ciphertext contains some padded values that generate a larger ciphertext. This increases the level of complexity of the encryption, and will most likely make a dictionary attack harder to succeed.

Once the message arrives on the recipient's side of the communication channel, the ciphertext gets decrypted using the private key in the following procedure: $m \equiv c^d \pmod{n}$, where c is the ciphertext.

4.3 A Practical Example

There are two actors in this scenario: Alice and Bob. Alice wants to send a message to Bob, and wants to cipher the message using RSA encryption. Bob picks two prime numbers, $p = 101$ and $q = 113$. The following variable is calculated using these two numbers, $n = p \cdot q = 11413$.

The next step is calculating the Phi (φ) value, so that $\varphi(n) = (p-1)\cdot(q-1)$. This gives us $\varphi(n) = 100 \cdot 112 = 11200$. Given the fact that $11200 = 2^6\ 5^2\ 7$, we can compute an integer e that will be used further as an exponent in the encryption process, as long as the exponent is not divisible by the numbers 2, 5 or 7. This point is verified as a part of the key generation process, where $(e, \varphi(n)) = 1$.

Bob picks an exponent number, say for example $e = 3$. Usually, it would be wise to pick a larger exponent, in terms of security, but we will use a small number in this scenario for the sake of simplicity. Using this exponent, Bob is able to generate the private key, such that $d = e^{-1} \bmod 11200 = 6597 = K_{private}$ (which is the secret key). Bob publishes the public key variables, n and e, which further can be used to encrypt the plaintext messages. The key generation process is complete and the message transmission process continues.

The next step is the actual encryption part, where the ciphertext is established using mathematics. In this scenario the message is ”USN”, which converts to the numbers ”85 83 78” when converting from text to decimal using the ASCII code table, which is shown in below figure:

Dec	Hex	Char
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9

Dec	Hex	Char
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z

Dec	Hex	Char
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z

Since 858378^3 is a very large number, the algorithm encrypts each character separately so that “U” is first encrypted, followed by “S” and then ”N” at last. Alice wants to send the ciphertext to Bob, and computes $c = 85^3 \bmod 11413 = 9236$. The first character has been encrypted, and the same process happens once again for the remaining characters of the plaintext.

$$\textbf{U}: 85^3 \bmod 11413 = \mathbf{9236}$$

$$\textbf{S}: 83^3 \bmod 11413 = \mathbf{1137}$$

$$\textbf{N}: 78^3 \bmod 11413 = \mathbf{6619}$$

The entire plaintext has been encrypted and the final ciphertext is **9236 1137 6619**. The ciphertext is sent to Bob and he decrypts the message using the same algorithm, followed by the same public key and the private key that Alice used to encrypt the message. Bob is only able to decrypt the message because he’s aware of the private key, otherwise it would not be able to convert the ciphertext back to plaintext in terms of cryptanalysis and brute force attacks.

Bob obviously knows the private key $K_{private}$ = 6597 and uses it to decrypt the message he received from Alice. The communication channel between Alice and Bob is aware of the fact that each block contains multiple digits, where each character of the plaintext is encrypted into a four-digit number in the ciphertext. Using the decryption formula, Bob computes $p = 9236^{6597} \bmod 11413 = 85$. The same process happens once again for the remaining blocks of the ciphertext, such that:

$$\mathbf{9236}: 9236^{6597} \bmod 11413 = 85 = \mathbf{U}$$

$$\mathbf{1137}: 1137^{6597} \bmod 11413 = 83 = \mathbf{S}$$

$$\mathbf{6619}: 6619^{6597} \bmod 11413 = 78 = \mathbf{N}$$

The ciphertext has successfully been decrypted and Bob is finally able to read the text. To summarize the steps of the message encryption and decryption process:

1. Bob chooses two prime numbers p and q, and computes $n = p \cdot q$

2. Bob chooses an exponent e

3. Bob shares n and e and makes these variables public, while p, q and d are kept secret.

4. Alice encrypts the message and sends the ciphertext to Bob.

5. Bob decrypts the ciphertext by using the private key, and reads the plaintext.

That is pretty much how the RSA algorithm operates. The security in this algorithm relies on the fact that this is a one-way function, and the only way of decrypting the ciphertext is with proper knowledge of the factorization $n = p \cdot q$. The ciphertext remains encrypted without this knowledge and there is really no way of decrypting it, as long as the prime numbers are large enough (as in at least 512 bits). That is why the RSA algorithm is considered one of the most secure and reliable algorithms as of today, and will hopefully remain this way for a long period of time.

5 RSA Implementation

We have developed the RSA algorithm in C++, where it encrypts and decrypts text messages using the different mathematical calculations. The program runs fine and it successfully encrypts and decrypts messages provided by the program user. The user writes pure text into the program console, without needing to manually convert the characters to decimal values first. The program handles all this work, so the user only needs to submit a text and nothing else. The user does however need to enter two prime numbers, along with the exponent value during the program launch.

We have decided to make this user-based and We did not want to hardcode these values because it would be easier to test the program with different prime numbers and exponent values if the user can decide these values during the program launch, which means that different prime numbers and exponent values can be used as well.

We also decided to encrypt each character at the time, instead of the entire plaintext in one batch. This was to prevent large numbers from causing a buffer overflow, and to handle and store each character one by one. All of the characters are stored inside a buffer (array), and these same values are loaded out of the buffer when the decryption process starts again. We believe this is a good way to handle large texts, as the program only cares about one character at a time, and does not care about how long the entire sentence is.

```
Enter two prime numbers (separated with whitespace): 13 19
Enter a value for exponent: 5
P = 13
q =19
n = 247
phi = 216
e =5
d= 173
Enter a text message: HELLO
Encrypted ciphertext: 113677105
Decrypted plaintext: HELLO

--------------------------------
Process exited after 11.81 seconds with return value 0
Press any key to continue . . .
```

We ran the program using these parameters. I used p = **13**, q = **19**, e = **5**, and encrypted the text "**HELLO**" which returned ciphertext "**113677105**". Afterwards the ciphertext was decrypted back to "**HELLO**".

5.1 Source Code:

```
/*-------------------------------------------------------------------
-----------     RSA ALGORITHM IMPLEMENTATION    ---------------------
--------------------------------------------------------------------*/

/*
  Subject: COMPUTER NETWORKS(2CP07)
```

```
   Reference of algorithm: Data Communications and Networking By Behrouz A.Forouzan(4th Editio
n)
   Code written by: Rajveer Rathod(19CP074)
   Group: 6
   Other group members: Valsal Dhupelia(19CP076)
                        Akshat Trivedi(19CP075)
   Purpose of Code: Demonstration of encryption and decryption of string using RSA Algorithm
   Platform: Windows 64-bit
*/

//Neccessary header files
#include <iostream>
#include <cmath>
#include <cstring>

// Alphabet values used for ASCII converstion:
static
const char Alphabet[26] = {
   'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I',
   'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R',
   'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'
   };

//structure
struct Primes
{
   double p, q;
};

//namespace to set private and public key
namespace Keys
{
   struct publicKey
   {
      double n, e;
   };

   struct privateKey
   {
      double d;
   };
};

//class
class RSA
{
   public:
      RSA(int, int);
```

```
    double gcd(double, double);
    void exponent();
    void privateKey();
    void convertToNumbers(char*);
    void encrypt();
    void decrypt();
    int modular_pow(int, int, int);
    void print();

  private:
    Primes primes;
    Keys::publicKey pubKey;
    Keys::privateKey privKey;

    // Array containing the plaintext/ciphertext:
    int *buffer = new int[512];
    int size = 0;
    double phi;
};

RSA::RSA(int _p, int _q)
{
  // Set the values for p and q
  primes.p = _p;
  primes.q = _q;

  // Set n = p *q
  pubKey.n = primes.p *primes.q;

  // Phi = (p-1) *(q-1)
  phi = (primes.p - 1) *(primes.q - 1);
}

double RSA::gcd(double a, double b)
{
  double temp;
  while (true)
  {
    temp = (int) a % (int) b;
    if (temp == 0)
    {
      return b;
    }

    a = b;
    b = temp;
  }
```

```
}

void RSA::exponent()
{
   // Set the exponent e:
   std::cout << "Enter a value for exponent: ";
   double _e;
   std::cin >> _e;

   //check whether the exponent is within given range or not
   if (_e < 1)
   {
      std::cout << "Exponent must be greater than 1.\n";
      exponent();
   }
   else if (_e > phi)
   {
      std::cout << "Exponent must be smaller than phi.\n";
      exit(1);
   }

   // Verify that e is valid:
   while (_e < phi)
   {
      if (gcd(_e, phi) == 1)
         break;
      else
          _e++;
   }

   pubKey.e = _e;
}

void RSA::privateKey()
{
   int i = 1;
   float temp;
   while (1)
   {
      temp = ((phi *i) + 1) / pubKey.e;
      //std::cout<<"inside the while, temp,i = "<<temp<<","<<i<<"\n";

      if (std::fmod(temp, 1) == 0)
      {
         //std::cout<<(int)temp<<"  "<<temp;
         privKey.d = (int) floor(temp);
         break;
      }
```

```
      i++;
    }
}


int RSA::modular_pow(int base, int exponent, int modulus)
{
    int result = 1;
    while (exponent > 0)
    {
        if (exponent % 2 == 1)
            result = (result *base) % modulus;
        exponent = exponent >> 1;
        base = (base *base) % modulus;
    }

    return result;
}

void RSA::encrypt()
{
    // Read the buffer array, encrypt each character:
    unsigned long long int ctext;
    std::cout << "Encrypted ciphertext: ";
    for (int i = 0; i < size; i++)
    {
        ctext = modular_pow(buffer[i], pubKey.e, pubKey.n);
        buffer[i] = ctext;
        std::cout << ctext;
    }

    std::cout << std::endl;
}

void RSA::decrypt()
{
    // Read the ciphertext from buffer array, decrypt each character:
    unsigned long long int ptext;
    std::cout << "Decrypted plaintext: ";
    for (int i = 0; i < size; i++)
    {
        ptext = modular_pow(buffer[i], privKey.d, pubKey.n);
        buffer[i] = ptext;
        std::cout << Alphabet[ptext];
    }

    std::cout << std::endl;
```

```
}

void RSA::convertToNumbers(char *plaintext)
{
   // Convert characters to ASCII decimals:
   for (int i = 0; i < strlen(plaintext); i++)
   {
      buffer[i] = plaintext[i] - 65;
   }

   size = strlen(plaintext);
}

void RSA::print()
{
   std::cout << "P = " << primes.p << "\n" << "q =" << primes.q << "\nn = "
   << pubKey.n << "\nphi = " << phi << "\ne =" << pubKey.e << "\nd= "
   << privKey.d << "\n";
}

int main()
{
   // Promt the user to enter two prime numbers:
   std::cout << "Enter two prime numbers (separated with whitespace): ";
   int p, q;
   std::cin >> p >> q;

   // Initiate the program, set the exponent and generate the private key:
   RSA rsa(p, q);
   rsa.exponent();
   rsa.privateKey();
   rsa.print();

   // Promt the user to enter a plaintext message:
   std::cout << "Enter a text message: ";
   char *plaintext = new char[512];
   std::cin >> plaintext;

   // Convert the plaintext characters into ASCII decimals:
   rsa.convertToNumbers(plaintext);

   // Run the encryption and decryption functions:
   rsa.encrypt();
   rsa.decrypt();
   return 0;
}
```

6 How Secure Is RSA?

1 in 172. That's the number of RSA public key certificates available through the internet that could be vulnerable to compromise due to shared cryptographic key factors.

These findings are according to a recent report on RSA certificate vulnerability from KeyFactor, a leading provider of secure digital identity management solutions and an established authority in the cybersecurity industry. A team of KeyFactor researchers presented their findings at the First IEEE Conference on Trust, Privacy, and Security in Intelligent Systems and Applications in December. The data indicates that due to improper random number generation, many RSA public keys are at risk of compromise because the researchers were able to use them to derive their private keys through a method known as "factoring."

Essentially, the research indicates that RSA is still secure, but many companies are implementing it in insecure ways. As such, it underscores the importance of organizations and manufacturers being "crypto agile" and adhering to cryptographic best practices to maintain trust and security.

This particular report on RSA certificate vulnerabilities, written by JD Kilgallin, states that the company collected and analysed 175 million RSA certificate public keys — 75 million they discovered on the internet, plus 100 million that were available through certificate transparency (CT) logs. They used a single Microsoft Azure cloud-hosted virtual machine and a greatest common divisor (GCD) algorithm for shared factors to conduct their analysis.

Here's what they discovered:

- Large numbers of RSA public keys can be collected through multiple sources and mined for common key factors.
- 1 in 172 certificates use keys that share a key factor with other certificates.
- They were able to crack nearly 250,000 distinct keys that correspond with 435,694 digital certificates.
- At least 435,000 weak certificate keys are vulnerable to "factoring" cyberattacks that exploit a key-related vulnerability.
- The majority of the vulnerable certificates were found on emerging IoT devices and network appliances.

The big takeaway here is that some IoT device manufacturers are using random number generators that lack strong entropy. It's more a matter of operator error than an actual weakness in the RSA algorithm itself. As a result of using random number generators (RNGs) with low entropy, they're generating prime numbers with poor randomness, which leads to the generation of private keys that can be compromised more easily.

At the most basic level, RSA public keys are the result of two large, randomly generated prime factors. They're created using random number generators. This means that the entire security premise of the RSA algorithm is based on using prime factorization as a method of one-way encryption. So, in other words, it's operating under the assumption that no one can determine two randomly-generated prime numbers within a reasonable amount of time — that no one can crack the encryption of an SSL/TLS certificate until long after it's replaced or expired.

Well, considering that it took a group of researchers more than 1,500 years of computing time (across hundreds of computers) to factor a 232-digit algorithm, that assumption seems plausible. But in reality, RSA is sometimes not as secure as we'd like it to be. It's not that RSA itself is insecure — it's that some companies implement it in a weak way.

That's because some random number generators aren't really that random. Furthermore, considering that the same RNGs are frequently used time and again, it reduces their effectiveness. If RSA public keys are generated with poor randomness, it means they could be vulnerable to a factoring cyberattack.

In this type of attack, cybercriminals collect large sums of public keys from the internet and analyse them to determine whether any two share the same factor. If two RSA moduli share one prime factor, it could result in a collision when applied to a large dataset. What this does is allow the actor to crack the corresponding private key.

All of this leads to this concern:

"As the number of keys grows, it is more likely that weakly generated factors in RSA public keys will be discovered. Coupled with the availability of cheap computing resources and sensitivity of communications, the attack is as potent as ever."

But there is a bit of light at the end of the tunnel.

7 Conclusion

The RSA algorithm is a very interesting cryptographic algorithm, and it is definitely one of the best and most secure algorithms available as of today. It provides great encryption and is reliable in terms of security and performance. The encryption security relies on the fact that the prime numbers used during the key generation process must be large enough to be unbreakable, and this is quite interesting. There might be a time in the future when super-computers are able to break these, but that would not be anytime soon at least.

Even though the algorithm provides great encryption and it is reliable, the overall security really relies on the program developer or the algorithm user. If the user picks small prime numbers, it compromises a lot of the security that the RSA algorithm provides, and therefore is very vulnerable to cryptanalysis and brute force attacks. So, in other words, just using the RSA algorithm is not enough - it must be used properly in order to gain the encryption level it initially provides, as it must be used correct in terms of the key generation process and the initial preparation of the algorithm.

We have learned incredibly much during this home exam period, and I never thought RSA would be this exciting, until I actually started reading about it. The mathematics behind the algorithm is incredible and I find it fascinating that such simple mathematical calculations can create such a large cryptographic algorithm, which is used worldwide.

We also appreciate the fact that we got the chance to actually code and implement the algorithm. It was a fun experience to use our programming skills to create an algorithm, and We did learn a lot both theoretically and practically. This assignment has overall been a great learning experience to us, and We hope you are satisfied with our work throughout this assignment!

8 References:

1. https://www.geeksforgeeks.org/rsa-algorithm-cryptography/
2. https://www.comparitech.com/blog/information-security/rsa-encryption/
3. Data Communications and Networking By Behrouz A.Forouzan:
4. https://www.brainkart.com/article/The-Security-of-RSA_8438/
5. https://www.educba.com/rsa-algorithm/
6. https://www.ijcsmc.com/docs/papers/June2013/V2I6201330.pdf
7. https://www.amsi.org.au/teacher_modules/pdfs/Maths_delivers/Encryption5.pdf
8. Totient Function -- from Wolfram MathWorld
9. Euclidean Algorithm -- from Wolfram MathWorld
10. Applications of Euler's theorem & totient function (johndcook.com)