CHAPTER ONE

1.0 INTRODUCTION

Since the beginning of time, man had secrets. These secrets he communicates with only a few, and the need arises that the secret remains secret–this was the beginning of cryptography: "the science of secrecy."

Two parties–Alice and Bob–wish to communicate with each other, and their means of communication is insecure because of an eavesdropper–Eve–who eavesdrops on their conversation. So they–Alice and Bob–adopt a technique to scramble their messages so it becomes gibberish and unreadable by Eve (except for them anyway). And with the use of a "key", the message can be unscrambled and returns to its original readable form. The entire process is called Cryptography.

1.1 BRIEF HISTORY

Cryptography, the use of codes and ciphers to protect secrets, is as old as man. Until recent decades, it has been the story of what might be called *classic cryptography* –that is, of methods of encryption that use pen and paper, or perhaps simple mechanical aids. In the early 20th century, the invention of complex mechanical and electromechanical machines, such as the Enigma rotor machine (used by the Germans during WWII), provided more sophisticated and efficient means of encryption; and the subsequent introduction of electronics and computing has allowed elaborate schemes of still greater complexity, most of which are entirely unsuited to pen and paper

The invention of the frequency-analysis technique for breaking monoalphabetic substitution ciphers, by Al-Kindi, an Arab mathematician, sometime around AD 800 proved to be the single most significant cryptanalytic advance until World War II. Al-Kindi wrote a book on cryptography entitled Risalah fi Istikhraj al-Mu'amma

(Manuscript for the Deciphering Cryptographic Messages), in which he described the first cryptanalytic techniques, including some for polyalphabetic ciphers , cipher classification, Arabic phonetics and syntax, and most importantly, gave the first descriptions on frequency analysis.

Essentially all ciphers remained vulnerable to the cryptanalytic technique of frequency analysis until the development of the polyalphabetic cipher, and many remained so thereafter. The polyalphabetic cipher was most clearly explained by Leon Battista Alberti around the year AD 1467, for which he was called the "father of Western cryptology". The French cryptographer Blaise de Vigenère devised a practical polyalphabetic system which bears his name, the Vigenère cipher.
In World War I the Admiralty's Room 40 broke German naval codes and played an important role in several naval engagements during the war, notably in detecting major German sorties into the North Sea that led to the battles of Dogger Bank and Jutland as the British fleet was sent out to intercept them. However its most important contribution was probably in decrypting the Zimmermann Telegram, a cable from the German Foreign Office sent via Washington to its ambassador Heinrich von Eckardt in Mexico which played a major part in bringing the United States into the war.

Mathematical methods proliferated in the period prior to World War II, notably in William F. Friedman's application of statistical techniques to cryptanalysis and cipher development and in Marian Rejewski's initial break into the German Army's version of the Enigma system in 1932.

By World War II, mechanical and electromechanical cipher machines were in wide use, although—where such machines were impractical—manual systems continued in use. The Germans made heavy use, in several variants, of an electromechanical rotor machine known as Enigma. Mathematician Marian Rejewski, at Poland's Cipher Bureau, in December 1932 deduced the detailed

structure of the German Army Enigma, using mathematics and limited documentation supplied by Captain Gustave Bertrand of French military intelligence. This was the greatest breakthrough in cryptanalysis in a thousand years and more, according to historian David Kahn.

Encryption in modern times is achieved by using algorithms that have a key to encrypt and decrypt information. These keys convert the messages and data into "digital gibberish" through encryption and then return them to the original form through decryption. In general, the longer the key is, the more difficult it is to crack the code. This holds true because deciphering an encrypted message by brute force would require the attacker to try every possible key. However, as technology advances, so does the quality of encryption. Since World War II, one of the most notable advances in the study of cryptography is the introduction of the asymmetric key ciphers (sometimes termed public-key ciphers). These are algorithms which use two mathematically related keys for encryption of the same message. Some of these algorithms permit publication of one of the keys, due to it being extremely difficult to determine one key simply from knowledge of the other. [Wikipedia].

1.2 STATEMENT OF PROBLEM

All previous ciphers were symmetric and this posed a problem of key distribution that is, the intended parties shared a common key for encryption and decryption, therefore they have to send this key to each other through a secure means–most a times a courier is used. If a need arise to change the key, the courier will have to send the new key to the intended receiver which will cost more resource and time.

Another aim is to establish a one-way-function, that is a function that is easy to compute and difficult to invert, except with the use of the privileged information–the trapdoor. This was the proposed cryptosystem by Diffe and Hellman at Stanford University, 1976. The RSA is thus one practical implementation of this asymmetric cryptosystem.

1.3 OBJECTIVE OF STUDY

The first aim is to design an asymmetric cryptosystem whereby one key is used to encrypt and an entirely different key is used for decryption. This can allow the intended receiver to publish one of his keys(the public key) and anyone would then encrypt their message before sending it across to the receiver who has the unique key for decryption. And in publishing the public key, the private key in the receiver's possession is untraceable from the information contained in the public key. Also to build a one-way-function using the underlying theorems in Mathematics–Number Theory in particular. Finally, a computer program will be designed to carryout the process of generating primes, key generation, encryption, and decryption. Reason being that the algorithm involves rigorous mathematical processes that are not suited to pen and paper. The language of interest is C++, since it has function and operators that will be of use.

1.4 SIGNIFICANCE OF STUDY

The concept of digital signature can be established with this proposed cryptosystem. People can therefore authenticate who sent a message without the sender being able to deny the claim. The one-way-function guarantees the security of the encryption scheme and then allows that the cipher is not susceptible to just any form of attack.

This once again shows that not all branch of Pure Mathematics are indeed "pure", for they have found interesting applications in the science of cryptography; such field as Number Theory has become a paradigm.

1.5 DEFINITION OF TERMS

- Encryption: the process of obscuring information to make it unreadable without special knowledge, key files, and/or passwords.

- Decryption: the process reversing an encryption, using algorithms to convert a cipher text into a plain text.

- Cipher: a crpytographic system using an algorithm that converts letters or sequence of bits to cipher text.

- Cipher text: the resulting data(text) after the original data has been encrypted.

- Plain text: this is the original data(text) before encryption, also the resulting data after decryption.

- Trapdoor: the special information that permits the inverse of a trapdoor function to be easily computed.

- Key: this is a piece of information(e.g. a passphrase) used to encode or decode a message.

- Public Key: this key is solely used to encrypt the data before sending across to the intended receiver

- Private Key: this key is used by the receiver to decrypt a message encrypted by the corresponding public key.

CHAPTER TWO

LITERATURE REVIEW

2.0 INTRODUCTION

A cryptosystem defines a pair of data transformations called encryption and decryption. Encryption is applied to the plain text i.e. the data to be communicated to produce cipher text i.e. encrypted data using encryption key. Decryption uses the decryption key to convert cipher text to plain text i.e. the original data. Now, if the encryption key and the decryption key is the same or one can be derived from the other then it is said to be symmetric cryptography. This type of cryptosystem can be easily broken if the key used to encrypt or decrypt can be found. To improve the protection mechanism Public Key Cryptosystem was introduced in 1976 by Whitfield Diffe and Martin Hellman of Stanford University. It uses a pair of related keys one for encryption and other for decryption. One key, which is called the private key, is kept secret and other one known as public key is disclosed. They also introduced digital signatures and attempted to apply number theory; their formulation used a shared secret key created from exponentiation of some number, modulo a prime numbers. However, they left open the problem of realizing a one-way function, possibly because the difficulty of factoring was not well studied at the time.

2.1 THE RSA

RSA is made of the initial letters of the surnames of Ron Rivest, Adi Shamir , and Leonard Adleman, who first publicly described the algorithm in 1978. Clifford Cocks, an English mathematician working for the UK intelligence agency GCHQ, had developed an equivalent system in 1973, but it was not declassified until 1997.

In August 1977, Rivest, Shamir, and Adleman provide four properties that the

encryption and decryption procedures that the RSA have:

1. Deciphering the enciphered form of a message M yields M. That is, $D(E(M)) = M$

2. E and D are easy to compute.

3. Publicly revealing E does not reveal an easy way to compute D. As such, only the user can decrypt messages which were encrypted with E. Likewise, only the user can compute D efficiently.

4. Deciphering a message M and then enciphering it results in M. That is, $E(D(M)) = M$

As Rivest, Shamir, and Adleman point out, if a procedure satisfying property (3) is used, it is extremely impractical for another user to try to decipher the message by trying all possible messages until they find one such that $E(M) = C$.

To test the security of RSA, Martin Gardner publishes Scientific American column about RSA in August 1977, including the $100 challenge (129 digit n) and the infamous "40 quadrillion years" estimate required to factor RSA-129=

114,381,625,757,888,867,669,235,779,976,146,61
2,010,218,296,721,242,362,562,561,842,935,706,
935,245,733,897,830,597,123,563,958,705,058,9
89,075,147,599,290,026,879,543,541 (129 digits)

RSA-129 was factored in 1994, using thousands of computers on Internet. "The magic words are squeamish ossifrage." Cheapest purchase of computing time ever! This gives credibility to difficulty of factoring, and helps establish key sizes needed for security.

No later than sooner was the Kid-RSA (KRSA), a simplified public-key cipher published in 1997, designed for educational purposes. Some people feel that learning Kid-RSA gives insight into RSA and other public-key ciphers, analogous to

simplified DES.

2.2 FACTORING THE RSA

Multiple polynomial quadratic sieve (MPQS) can be used to factor the public modulus n(one of the public key component). The time taken to factor 128-bit and 256-bit n on a desktop computer (Processor: Intel Dual-Core i7-4500U 1.80GHz) are respectively 2 seconds and 35 minutes.

In 2009, Benjamin Moody factored an RSA-512 bit key in 73 days using only public software (GGNFS) and his desktop computer (dual-core Athlon64 at 1,900 MHz). Just under 5 gigabytes of disk was required and about 2.5 gigabytes of RAM for the sieving process. The first RSA-512 factorization in 1999 required the equivalent of 8,400 MIPS years over an elapsed time of about 7 months.

As of 2010, the largest factored RSA number was 768 bits long (232 decimal digits, see RSA-768). Its factorization, by a state-of-the-art distributed implementation, took around fifteen hundred CPU years (two years of real time, on many hundreds of computers). No larger RSA key is publicly known to have been factored. In practice, RSA keys are typically 1024 to 4096 bits long. Some experts believe that 1024-bit keys may become breakable in the near future or may already be breakable by a sufficiently well-funded attacker (though this is disputed); few see any way that 4096-bit keys could be broken in the foreseeable future. Therefore, it is generally presumed that RSA is secure if n is sufficiently large. If n is 300 bits or shorter, it can be factored in a few hours on a personal computer, using software already freely available. Keys of 512 bits have been shown to be practically breakable in 1999 when RSA-155 was factored by using several hundred computers and are now factored in a few weeks using common hardware.

In 1994, Peter Shor showed that a quantum computer (if one could ever be practically created for the purpose) would be able to factor in polynomial time(unlike other factoring algorithms whose computational time has an exponential growth), breaking RSA. Michael J. Wiener showed that if p(one of the primes that make up n= p.q) is between q and 2q (which is quite typical) and $d < n^{1/4}/3$, then d can be computed efficiently from n and e .

2.3 ATTACKS ON RSA

RSA has the property that the product of two ciphertexts equals to the encryption of the product of the respective plaintexts. That is $M_1^e.M_2^e \equiv (M_1.M_2)^e \pmod{n}$. Because of this multiplicative property a chosen-ciphertext attack is possible. E.g., an attacker, who wants to know the decryption of a ciphertext $c \equiv m^e \pmod{n}$ may ask the holder of the private key d to decrypt an unsuspicious-looking ciphertext $C' \equiv Cr^e \pmod{n}$ for some value r chosen by the attacker. Because of the multiplicative property C′ is the encryption of M.r (mod n). Hence, if the attacker is successful with the attack, he will learn M.r (mod n) from which he can derive the message m by multiplying M.r with the modular inverse of r modulo n.

Timing attacks: Kocher described a new attack on RSA in 1995: if the attacker Eve knows Alice's hardware in sufficient detail and is able to measure the decryption times for several known ciphertexts, she can deduce the decryption key d quickly. This attack can also be applied against the RSA signature scheme. In 2003, Boneh and Brumley demonstrated a more practical attack capable of recovering RSA factorizations over a network connection (e.g., from a Secure Sockets Layer (SSL)-enabled webserver). This attack takes advantage of information leaked by the Chinese remainder theorem optimization used by many RSA implementations.

Bell laboratories discovered that all tamperproof devices of cryptosystems, which use public key cryptography for user authentication without special countermeasure, are at the risk of the occurrence of hardware faults. For example,

smart cards that are used for data storage, cards that personalize cellular phones, cards that generate digital signatures or authenticate users for remote login to corporate networks are all vulnerable to this attack.

Owing to the fact that RSA encryption is a deterministic encryption algorithm (i.e., has no random component) an attacker can successfully launch a chosen plaintext attack against the cryptosystem, by encrypting likely plaintexts under the public key and test if they are equal to the ciphertext. A cryptosystem is called semantically secure if an attacker cannot distinguish two encryptions from each other even if the attacker knows (or has chosen) the corresponding plaintexts.

Padding schemes: To avoid these problems, practical RSA implementations typically embed some form of structured, randomized padding into the value M(the integer assigned to the block of data) before encrypting it. This padding ensures that m does not fall into the range of insecure plaintexts, and that a given message, once padded, will encrypt to one of a large number of different possible ciphertexts. However, at Eurocrypt 2000, Coron et al. showed that for some types of messages, this padding does not provide a high enough level of security. Furthermore, at Crypto 1998, Bleichenbacher showed that this version is vulnerable to a practical adaptive chosen ciphertext attack.

Adaptive chosen ciphertext attacks: In 1998, Daniel Bleichenbacher described the first practical adaptive chosen ciphertext attack, against RSA-encrypted messages using the PKCS#1 v1 padding scheme (a padding scheme randomizes and adds structure to an RSA- encrypted message, so it is possible to determine whether a decrypted message is valid).

2.4 OTHER PKC PROPOSALS

1978: Merkle/Hellman (knapsack)

1979: Rabin/Williams (factoring)

1984:Goldwasser/Micali(QR)

1985: El Gamal(DLP)

1985: Miller/Koblitz(elliptic curves)

1998: Cramer/Shoup

CHAPTER THREE

THE RSA CRYPTOSYSTEM

3.1 INTRODUCTION

The RSA cryptosystem is built upon some underlying theorems in number theory such as Fermat's little theorem, Euler's phi-function among others. Before moving into the algorithm, we first take a look at these theorems (mathematical-background) that serve as the building blocks for the security of the RSA.

3.2 THEOREMS

THEOREM 1 (Fermat's Little Theorem): Let p be a prime, and suppose that $p \nmid a$. Then $a^{p-1} \equiv 1 \ (mod\ p)$.

Proof:

We begin by considering the first p-1 positive multiples of a; that is the integers

$$a, 2a, 3a, \dots\dots, (p-1)a$$

None of these numbers is congruent modulo p to any other, nor is any congruent to zero. Indeed if it happened that

$$ra \equiv sa \ (mod\ p) \qquad 1 \leq r < s \leq p-1$$

then a could be canceled to give $r \equiv s \ (mod\ p)$, which is impossible (take note of the interval above). Then the previous set of integers $\{a, 2a, 3a, \dots\dots, (p-1)a\}$ must be congruent modulo $p\ to$ 1,2,3, $\dots\dots, p-1$ (taken in some order). Multiplying all these congruencies together, we find that

$$a.2a.3a.\dots\dots.(p-1)a \equiv 1.2.3.\dots\dots.(p-1) \ (mod\ p)$$

therefore

$$a^{p-1}.(p-1)! \equiv (p-1)! \ (mod\ p)$$

Once $(p-1)!$ is canceled from both sides of the preceding congruence (this is possible because since p does not divide $(p-1)!$), the line of reasoning culminates in the statement that:

$$a^{p-1} \equiv 1 \ (mod\ n),$$ ■

which is Fermat's little theorem.

DEFINITION 1 (Number-theoretic function): A number theoretic function f defined on the set of natural numbers is a real or complex valued function that expresses some arithmetical property of some natural number.

DEFINITION 2 (Multiplicative Property): A number-theoretic function f is said to be multiplicative if

$f(mn) = f(m).f(n)$ whenever $gcd(m,n) = 1$ [by gcd we mean the greatest common divisor]. Since

$f(n) = f(n.1) = f(n).f(1)$

where f is a multiplicative function, it follows from the last equation that

$f(1) = 1,$

for all multiplicative functions.

THEOREM 2 (Euler's Phi-function): if p is prime and $k > 0$, then $\phi(p^k) = p^k - p^{k-1} = p^k(1 - \frac{1}{p})$.

[The Euler's Phi-function is a number theoretic function that outputs the total number of natural numbers less than p^k and are relatively prime to it.]

Proof:

Clearly $gcd(n, p^k) = 1$ if and only if p does not divide n. There are p^{k-1} integers between 1 and p^k divisible by p, namely,

$p, 2p, 3p, \ldots\ldots, sp, p^k$

Observe that in the sequence the common difference is

$T_n - T_{n-1} = p$

therefore,

$p^k - sp = p$

and

$s = p^{k-1} - 1$

Also, the sequence is such that if

$ap, bp, cp, \ldots\ldots$

then,

$$b = a + 1, \qquad c = b + 1$$

The resulting sequence is thus,

$$p, 2p, 3p, \ldots\ldots, (p^{k-1} - 1)p, (p^{k-1})p$$

Thus the set $\{1, 2, 3, \ldots\ldots, p^k\}$ contains exactly $p^k - p^{k-1}$ integers that are relatively prime to p^k. So by definition of the phi-function $\phi(p^k) = p^k - p^{k-1}$. ■

The Euler's phi-function is one number-theoretic function with the multiplicative property. Therefore,

$$\phi(mn) = \phi(m).\phi(n), \qquad where\ gcd(m, n) = 1$$

By the fundamental theorem of arithmetic, every natural number can be uniquely expressed as a product of primes; that is

$$n = {p_1}^{k_1}.{p_2}^{k_2}.\ldots\ldots{p_r}^{k_r}$$

therefore,

$$n = \prod_{i=1}^{r} {p_i}^{k_i}$$

where p_i is prime and k_i, n are natural numbers.

Therefore, for any natural number n, the total number of positive integers less than n and relatively prime to it is given by

$$\phi(n) = \phi({p_1}^{k_1}.{p_2}^{k_2}.\ldots\ldots{p_r}^{k_r})$$

$$= \phi({p_1}^{k_1}).\phi({p_2}^{k_2}).\ldots\ldots\phi({p_r}^{k_r})$$

$$= {p_1}^{k_1}\left(1 - \frac{1}{p_1}\right).{p_2}^{k_2}\left(1 - \frac{1}{p_1}\right).\ldots.{p_r}^{k_r}\left(1 - \frac{1}{p_r}\right)$$

$$= {p_1}^{k_1}.{p_2}^{k_2}.\ldots\ldots{p_r}^{k_r}.\left(1 - \frac{1}{p_1}\right).\left(1 - \frac{1}{p_1}\right).\ldots.\left(1 - \frac{1}{p_r}\right)$$

$$= n.\prod_{i=1}^{r}\left(1 - \frac{1}{p_i}\right) \qquad ■$$

The above is considered the Euler's phi-function.

COROLLARY 1: If n is prime, then $\phi(n) = n - 1$.

THEOREM 3 (Euler's generalization of the Fermat's little theorem):

Given $gcd(a,n) = 1, \quad a^{\phi(n)} \equiv 1 \ (mod\ n)$.

Proof:

Consider the set of all numbers less than n and relatively prime to it. Let $\{c_1, c_2, \dots, c_{\phi(n)}\}$ be this set.

Consider a number a where $(a < n)$, relatively prime to n, that is $a \in \{c_1, c_2, \dots, c_{\phi(n)}\}$.

First observe that for any c_i,

$$a.c_i \equiv c_j \ (mod\ n) \quad for\ some\ j.$$

This is evidently true since a and c_i are themselves relatively prime to n, their product has to be relatively prime to n.

And if

$$a.c_i \equiv a.c_j \ (mod\ n),$$

then

$$c_i \equiv c_j \ (mod\ n).$$

This is possible since a is relatively prime to n. Hence if we now consider the set

$$\{a.c_1, a.c_2, \dots, a.c_{\phi(n)}\},$$

this is just a permutation of

$$\{c_1, c_2, \dots, c_{\phi(n)}\}.$$

Thereby, we have

$$a.c_1.a.c_2.\dots.a.c_{\phi(n)} \equiv c_1.c_2.\dots.c_{\phi(n)} \ (mod\ n)$$

$$a^{\phi(n)}.\prod_{i=1}^{\phi(n)} c_i \equiv \prod_{i=1}^{\phi(n)} c_i \ (mod\ n)$$

We note that $\prod_{i=1}^{\phi(n)} c_i$ is relatively prime to n; hence it cancels out to leave us with

$$a^{\phi(n)} \equiv 1 \ (mod\ n) \qquad \blacksquare$$

3.3 THE RSA ALGORITHM

The RSA algorithm is rooted in the Euler's theorem(theorem 3). If we wish to encrypt a message M (where M is the numerical value of this message), we start by selecting two large primes (about 200 digits long for practical purpose) p and q. The prime numbers should be randomly generated ; this can be done using a pseudo-random-number generator (PRNG function) in any programming language. The Fermat's little theorem (theorem 1) can then be used to test for primality, that is, if r where the generated random number, we check for

$$2^r \equiv b \ (mod\ r). \qquad \textbf{[equation 3.1]}$$

If b = 2, then r must be prime because by theorem 1, if given p (a prime number)

$$a^{p-1} \equiv 1 \ (mod\ p)$$

there is no harm in multiplying both sides of the above equation by a to give us

$$a^p \equiv a \ (mod\ p).$$

Therefore using equation 1 we can always test if the random number generated is prime.

Next, we compute n—the product of p and q; then we compute $\phi(n) = (p-1)(q-1)$. We proceed to choose a component of our secret key, 'e', such that $gcd(e, \phi(n)) = 1 \ and \ 1 < e < \phi(n)$. (e, n) will serve as the public encryption key. We will use this to compute another component the private(decryption) key, d, such that

$$e.d \equiv 1 \ (mod\ \phi(n)). \qquad \textbf{[equation 3.2]}$$

That is d is multiplicative inverse of e modulo $\phi(n)$. The private key is thus (d,n). To encrypt a message M with the public key, we take M ($M < n$) and raise it to e modulo n. That is

$$M^e \equiv C \ (mod\ n)$$

where C is the residue of the exponentiation. The same process follows for decryption, C is raised to d modulo n, and the original message is retrieved. That is

$$C^d \equiv M \ (mod\ n)$$

$$(M^e)^d \equiv M \ (mod\ n)$$

$$M^{ed} \equiv M\ (mod\ n) \qquad \textbf{[equation 3.3]}$$

This is true because by theorem 3

$$M^{\ \Phi(n)} \equiv 1\ (mod\ n) \quad \text{Where } n = p.q \quad \text{and} \quad \phi(n) = (p-1)(q-1)$$

then,

$$M^{\ \Phi(n).k} \equiv 1^k\ (mod\ n)$$

$$M^{\ \Phi(n).k} \equiv 1\ (mod\ n)$$

$$M^{\ \Phi(n).k}.M \equiv 1.M\ (mod\ n)$$

$$M^{\ \Phi(n).k+1} \equiv M\ (mod\ n) \qquad \textbf{[equation 3.4]}$$

Following from equation 3.3 and 3.4, we have that

$$e.d = \phi(n).k + 1$$

then,

$$e.d - 1 = \phi(n).k \qquad \text{or} \qquad e.d - k.\phi(n) = 1 \qquad \textbf{[equation 3.5]}$$

which implies that

$$e.d \equiv 1\ (mod\ \phi(n)) \qquad \textbf{[equation 3.6]}$$

which was already suggested in equation 3.2. The value for d can be obtained by solving equation 3.5 which is a simple Diophantine equation. This can be solved using the Extended Euclid's algorithm. The Extended Euclid's algorithm can be bypassed by applying theorem 3 as follows.

$$e^{\phi(u)} \equiv 1\ (mod\ u)$$

therefore

$$e^{\phi(u)} \equiv e^{-1}(mod\ u) \qquad \textbf{[equation 3.7]}$$

As earlier said that d is the multiplicative inverse of e modulo ϕ(n), in other words $d = e^{-1}$. Therefore following from equation 3.7, we have that

$$d \equiv e^{\phi(u)-1}\ (mod\ u) \qquad \textbf{[equation 3.8]}$$

Note that $u = \phi(n)$. So with equation 3.8 we can compute the decryption key d.

3.4 USING THE RSA FOR DIGITAL SIGNATURE

As proposed by Diffe and Hellman, the public key cryptosystem can serve as a means of appending signatures to electronic mail. The RSA is one that allows for an easy implementation of the concept of digital signature.

Suppose Alice and Bob were to send mails to each other, and for the purpose of authentication, Alice wishes to append her signature on the mail before sending across to Bob. To do this, she proceeds as follows

1. She acquires Bob's public key (e_B, n_B), together with her private key (d_A, n_A) to sign a document M.
2. She first encrypts the document with her private key

   $M^{d_A} \equiv C_1 \ (mod\ n_A)$
3. Next she encrypts the result with Bob's public key

   ${C_1}^{e_B} \equiv C_2 \ (mod\ n_B)$
4. She the sends C_2 to Bob; then Bob proceeds to verify that the document came from Alice (as claimed). He starts by decrypting C_2 with his private key (d_B, n_B)

   ${C_2}^{d_B} \equiv C_1 \ (mod\ n_B)$
5. Then he uses Alice's public key (e_A, n_A) to encrypt the result (decrypting is actually what is done here)

   ${C_1}^{e_A} \equiv M \ (mod\ n_A)$

The result of the entire process is M, as it can be observed that

$$[[[M^{d_A} \ (mod\ n_A)]^{e_A} \ (mod\ n_B)]^{d_B} \ (mod\ n_B)]^{e_A} \ (mod\ n_A) \ \equiv \ M \ (mod\ n_A)$$

Since $e_B$and $d_B$are multiplicative inverse of each other, their product is 1 (same happens for e_A and d_A). The entire process leaves us with M.

Bob is certain that the mail came from Alice since it was her public key that assisted in unlocking the document that was encrypted with her private key.

CHAPTER FOUR

THE PROCESS OF ENCRYPTION AND DIGITAL SIGNATURE

4.0 A TEXT ENCRYPTION USING THE RSA

Having considered the RSA algorithm in the previous chapter, we now proceed to use it for its designed purpose—encryption—by following the already outlined rules for a successful encryption.

4.0.0 KEY GENERATION

First we select two random numbers and test for primality. The numbers chosen will be 83 and 109. We see that both numbers are prime because

$$2^{83} \equiv 2\ (mod\ 83) \qquad and \qquad 2^{109} \equiv 2\ (mod\ 109)$$

The last results are due to equation 3.1 .For practical purpose, it is advised that one should select primes (of varying length) as large as 200 digits so to beat the current speed of the modern computers, also the most efficient factoring algorithms.

Next we compute the product of the primes and its corresponding output for the phi-function.

$$n = p.q = 83 \times 109 = 9047$$

$$\phi(9047) = (83-1).(109-1) = 8856$$

Now we choose an encrypting exponent (public key), e, that is relatively prime to $\phi(9047)$ and is an integer in the open interval $(1\ , \phi(9047))$. Choosing $e = 11$, since $gcd(11,8856) = 1$. Then we proceed to compute the decrypting exponent (private key), d—the multiplicative inverse of e modulo 8856.

$$11d \equiv 1\ (mod\ 8856)$$

Then by equation 3.8,

$$d \equiv 11^{\phi(8856)\,-\,1}\ (mod\ 8856)$$

Since,

$$8856 = 2^3.3^3.41$$

then

$$\phi(8856) = 8856.(1-\tfrac{1}{2}).(1-\tfrac{1}{3}).(1-\tfrac{1}{41}) = 8856.{}^{1}/_{2}\,.{}^{2}/_{3}\,.{}^{40}/_{41} = 2880$$

So we have d to be,

$$d = 11^{2880-1} \ (mod\ 8856) \ = \ 11^{2879} (mod\ 8856) \ \equiv \ 8051 \ (mod\ 8856)$$

We now have $d = 8051 \ (mod\ 8856)$.

On a side note, the encrypting exponent should be chosen to be quite small, so that the corresponding inverse—d—will be sufficiently large. We now have the following pair as our public and private key respectively (11,9047), (8051,9047). The primes—83 and 109—with our result for $\phi(9047) = 8856$ should be kept secret, for with either of them, our private key—d—can easily be computed for given the public key—e.

4.0.1 NUMERICAL REPRESENTATION

Now we adopt a scheme that assigns a numerical value to each character that will be used in this project work. The characters will involve all printable ASCII (American Standard Code for Information Interchange) characters (and some UNICODE characters, just to complete a total of 100 characters), as follows

!	"	#	$	%	&	'	(	)	*	+	‘
01	02	03	04	05	06	07	08	09	10	11	12
-	.	/	0	1	2	3	4	5	6	7	8
13	14	15	16	17	18	19	20	21	22	23	24
9	:	;	<	=	>	?	A	B	C	D	E
25	26	27	28	29	30	31	32	33	34	35	36
F	G	H	I	J	K	L	M	N	O	P	Q
37	38	39	40	41	42	43	44	45	46	47	48
R	S	T	U	V	W	X	Y	Z	[	\	]
49	50	51	52	53	54	55	56	57	58	59	60
^	_	`	a	b	c	d	e	f	g	h	I
61	62	63	64	65	66	67	68	69	70	71	72
j	k	l	m	n	o	p	q	r	s	t	u
73	74	75	76	77	78	79	80	81	82	83	84

v	w	x	y	z	{	|	}	~	^	¬	˜
85	86	87	88	89	90	91	92	00	93	94	95
o	2	3	1								
96	97	98	99								

The character ~ will be adopted as space in our encryption scheme.

[The plaintext and the private key component will be appear in red in this project work, while the ciphertext and the public key component will appear in blue.]

4.0.2 ENCRYPTION

It should be ensured that the numerical values assigned are strictly less than the modulo n. Now we proceed to encrypt a text with the RSA algorithm using the keys just generated. The text to be encrypted is the Bible passage Psalm 91: 1-2

He~who~dwells~in~the~secret~place~of~the~Most~High~shall~abide~under~the~shadow~of~the~Almighty.~I~will~say~of~the~Lord,~"He~is~my~refuge~and~my~fortress;~my~God,~in~Him~I~will~trust."~(Psalm~91:1-2)

To reduce the total number of computation needed to encrypt the above text, the characters of the text will be picked in "twos", that is

He ~w ho ~d we ll

From the array of characters in the previous page, we have the corresponding numerical values for the last line of the selected characters as

He: 3968 ~w: 0086 ho: 7178 ~d: 0067 we: 8668 ll: 7575

Before proceeding to encrypt, we bear in mind that the process of exponentiation will produce values that can be very large such that the computer will not be able to hold these values before their residue is computed for. For this reason, the

computation can be done using repeated squaring and multiplication.

$3968^{11} \pmod{9047} = (3968^2)^5 \cdot 3968 \equiv_{9047} 3244^5 \cdot 3968$

$= (3244^2)^2 \cdot 3244 \cdot 396 \equiv_{9047} 1875^2 \cdot 3244 \cdot 3968 \equiv_{9047} 5389 \cdot 3244 \cdot 3968$

$\equiv_{9047} 5389 \cdot 7358 \equiv 8308 \pmod{9047}$

$86^{11} \pmod{9047} = (86^2)^5 \cdot 86 \equiv_{9047} 7396^5 \cdot 86 = (7396^2)^2 \cdot 7396 \cdot 86$

$\equiv_{9047} 2654^2 \cdot 636056 \equiv_{9047} 5150 \cdot 636056 \equiv 4922 \pmod{9047}$

$7178^{11} \pmod{9047} = (7178^2)^5 \cdot 7178 \equiv_{9047} 1019^5 \cdot 7178$

$= (1019^2)^2 \cdot 1019 \cdot 7178 \equiv_{9047} 7003^2 \cdot 7314382 \equiv_{9047} 7269 \cdot 4406$

$\equiv 0834 \pmod{9047}$

$67^{11} \pmod{9047} = (67^2)^5 \cdot 67 \equiv_{9047} 4489^5 \cdot 67 = (4489^2)^2 \cdot 4489 \cdot 67$

$\equiv_{9047} 3452^2 \cdot 300763 \equiv_{9047} 1405 \cdot 2212 \equiv 4739 \pmod{9047}$

$8668^{11} \pmod{9047} = (8668^2)^5 \cdot 8668 \equiv_{9047} 7936^5 \cdot 8668$

$= (7936^2)^2 \cdot 7936 \cdot 8668 \equiv_{9047} 3929^2 \cdot 4907 \equiv_{9047} 2859 \cdot 4907$

$\equiv 6263 \pmod{9047}$

$7575^{11} \pmod{9047} = (7575^2)^5 \cdot 7575 \equiv_{9047} 4551^5 \cdot 7575$

$= (4551^2)^2 \cdot 4551 \cdot 7575 \equiv_{9047} 3018^2 \cdot 4755 \equiv_{9047} 7042 \cdot 42755$

$\equiv 1763 \pmod{9047}$

We now proceed to split the numbers in “twos”

8308	83	08	4922	49	22	0834	08	34
	T	(		R	6		(	C
4739	47	39	6263	62	63	1763	17	63
	P	H		_	`		1	`

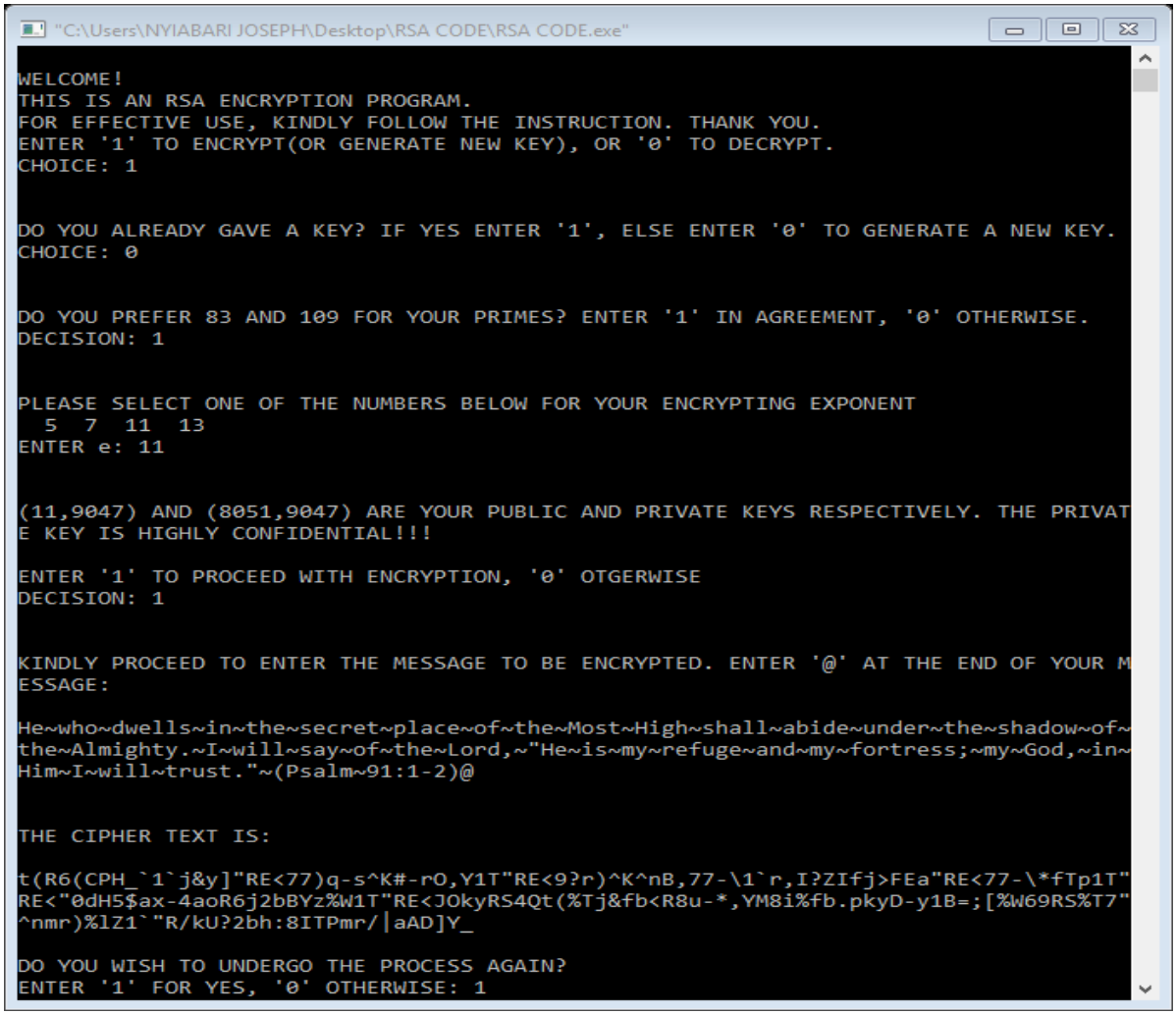
Table 4.1.

The ciphertext obtained from the given plaintext is: t(R6(CPH_`1

If we continue in the trend as before, the entire text will be encrypted into

t(R6(CPH_`1`j&y]"RE<77)q-s^K#-rO,Y1T"RE<9?r)^K^nB,77-\1`r,I?ZIfj>FEa"RE<77-\*fTp1T"RE<"0dH5$ax-4aoR6j2bBYz%W1T"RE<JOkyRS4Qt(%Tj&fb<R8u-*,YM8i%fb.pkyD-y1B=;[%W69RS%T7"^nmr)%lZ1`"R/kU?2bh:8ITPmr/|aAD]Y_

Below is a screenshot of the encryption process, using the computer program designed for this purpose.



4.0.3 DECRYPTION

Having finished with the process of encryption, we commence now to decrypt the encrypted text. Referring to table 4.1, we have the numerical equivalent of t(R6(CPH_`1 to be 830849220834473962631763.

The decryption process follows the same procedure as the encryption

$8308^{8051} \pmod{9047} = 8308^{4428} \cdot 8308^{3623} \equiv_{9047} 1 \cdot 8308^{3623} \equiv 3968 \pmod{9047}$

$4922^{8051} \pmod{9047} = 4922^{4428} \cdot 4922^{3623} \equiv_{9047} 1 \cdot 4922^{3623} \equiv 0086 \pmod{9047}$

$0834^{8051} \pmod{9047} = 834^{4428} \cdot 834^{3623} \equiv_{9047} 1 \cdot 834^{3623} \equiv 7178 \pmod{9047}$

$4739^{8051} \pmod{9047} = 4739^{4428} \cdot 4739^{3623} \equiv_{9047} 1 \cdot 4739^{3623} \equiv 0067 \pmod{9047}$

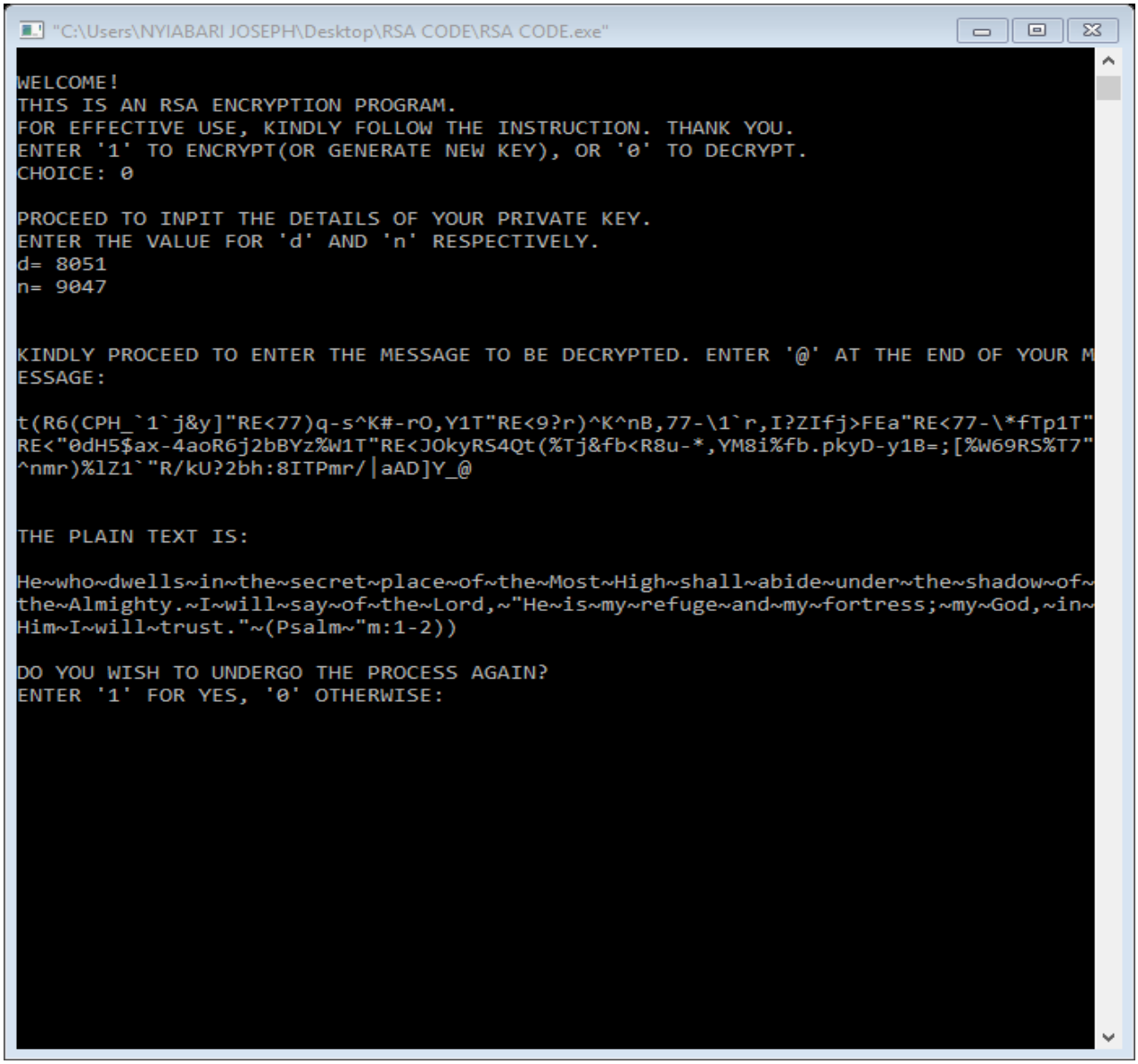
$6263^{8051} \pmod{9047} = 6263^{4428} \cdot 6263^{3623} \equiv_{9047} 1 \cdot 6263^{3623} \equiv 8668 \pmod{9047}$

$1763^{8051} \pmod{9047} = 1763^{4428} \cdot 1763^{3623} \equiv_{9047} 1 \cdot 1763^{3623} \equiv 7575 \pmod{9047}$

The results above are what we initially commenced with, few pages back. The decrypted text (plaintext) is therefore He~who~dwell. And if the computation for the other portion of the ciphertext continues, the result for the entire plaintext will be

He~who~dwells~in~the~secret~place~of~the~Most~High~shall~abide~under~the~shadow~of~the~Almighty.~I~will~say~of~the~Lord,~"He~is~my~refuge~and~my~fortress;~my~God,~in~Him~I~will~trust."~(Psalm~"m:1-2))

Below is a screenshot of the decryption process, using the computer program designed for this purpose



.

As one would clearly see this one way function guarantees security, since finding the eth-root of C to obtain M given

$$M^e \equiv C \ (mod\ n)$$

is one such task that can be solved in a polynomial time.

4.1 DIGITAL SIGNATURE USING THE RSA

Assuming two parties—Alice and Bob—wish to involve in digital signature, they both must have a public and private key pair. Following the procedures already highlighted in the previous chapters, we proceed to digitally sign and authenticate the document whose numerical value is 24. The following key pairs are for Alice and Bob respectively $(e_A = 5, d_A = 29, n_A = 91), (e_B = 7, d_B = 19, n_B = 69)$. Before Alice signs the document to be sent to Bob, she first obtains Bobs public key from her directory $(e_B = 7,\ n_B = 69)$; using this and her private key $(d_A = 29, n_A = 91)$ she proceeds to sign the document. First using her private key

$$24^{29} \pmod{91} = (24^2)^{14} \cdot 24 \equiv_{91} 30^{14} \cdot 24 = (30^2)^7 \cdot 24 \equiv_{91} 81^7 \cdot 24 = (81^2)^3 \cdot 81 \cdot 24$$

$$\equiv_{91} 9^3 \cdot 81 \cdot 24 \equiv_{91} 1 \cdot 81 \cdot 24 = 1944 \equiv 33 \pmod{91}$$

Now she encrypts the result using Bob's public key

$$33^7 \pmod{69} = (33^2)^3 \cdot 33 \equiv_{69} 54^3 \cdot 33 \equiv_{69} 6 \cdot 33 = 198 \equiv 60 \pmod{69}$$

Now Alice sends this value (60 (mod 69)) to Bob; on receiving this he proceeds to authenticate who it came from—Alice in our case. Bob starts by decrypting the message using his private key

$$60^{19} \pmod{69} = (60^2)^9 \cdot 60 \equiv_{69} 12^9 \cdot 60 = (12^2)^4 \cdot 12 \cdot 60 \equiv_{69} 6^4 \cdot 12 \cdot 60$$

$$\equiv_{69} 54 \cdot 12 \cdot 60 = 54 \cdot 720 \equiv_{69} 54 \cdot 30 = 1620 \equiv 33 \pmod{69}$$

Then he further decrypts the result using Alice's public key

$$33^5 \equiv 24 \pmod{91}$$

Bob has obtained the same result that Alice originally sent. This allows him verify where the mail came from—Alice—because if the mail never came from her, then her public key would never have decrypted the document to its original content.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.0 CONCLUSION

Having seen the implementation of the RSA cryptosystem, it is evident that the factoring problem served as our one-way-function, and the key exchange problem has been settled. Still the RSA is susceptible to some forms of attack (though rare) as discussed in chapter 2, this is why there are continuous improvements on some aspects of the algorithm. Some improvements includes the reduction of the keyspace, that is d and e, using the Carmichael's function

$$\lambda(n) = lcm(p-1, q-1)$$

instead of the Euler's phi-function. $\lambda(n)$ is the least positive exponent that guarantees the equation below to be true

$$a^{\lambda(n)} \equiv 1 \ (mod\ n)$$

Since $\phi(n)$ is a multiple of $\lambda(n)$, we see that this still satisfies theorem 3

$$a^{\phi(n)} \equiv a^{t.\lambda(n)} \ (mod\ n) \qquad t \in \boldsymbol{Z}$$

There are other techniques in implementing the RSA, one such technique is the CRT (Chinese Remainder Theorem) method. This allows the user the flexibility of selecting 3 (instead of the conventional 2) random primes Also using the CRT reduces the magnitude of the computation to be carried out by the computer, therefore the throughput is low and the power consumption greatly reduced.

For more reliable security, it is suggested that n is 200 digits long. Though an 80-digit n provides moderate security against an attack using current technology; using 200 digits provides a margin of safety against future development. Longer or shorter lengths can be used depending on the relative importance of encryption speed and security in the application at hand.

Asides the relative difficulty of factoring, the eth-root of some number modulo n is also conjectured to be relatively difficult to compute. For there is no efficient algorithm to compute

$$C^{1/e} \equiv M \ (mod\ n)$$

if given

$$M^e \equiv C(mod\ n)$$

5.1 RECOMMENDATIONS

I humbly conclude by suggesting that the implementation of the RSA be implemented by another scholar using the CRT algorithm, because of the many advantages it poses. In addition, the Carmichael's function ($\lambda(n)$) should be used in place of the phi-function ($\phi(n)$). Lastly, the task of refuting the original authors claim that the eth-root is relatively infeasible to compute and that with the current technology and the most efficient factoring algorithm, some very large number—say n—about 200 digits long, is difficult to factor (split into its component primes), is still open to the general public.

REFERENCES

APPENDIX

PROGRAM SOURCE CODE

```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>

using namespace std;

int choosePrime();
unsigned long long GetRand();
unsigned long long ModExp( unsigned long long,unsigned long long,unsigned long
long);
int gcd(int,int);
int InvF(int,int);
long assignNum(char);
char assignChar(int);
void split(long,long &,long &);




int main()
{
   int response;
do
{
        cout <<"\nWELCOME!\nTHIS IS AN RSA ENCRYPTION PROGRAM.\nFOR
EFFECTIVE USE, KINDLY FOLLOW THE INSTRUCTION. THANK YOU.\nENTER '1'
TO ENCRYPT(OR GENERATE NEW KEY), OR '0' TO DECRYPT.\nCHOICE: ";
   int /* p[2],*/N;
   long const p[2]= {83,109};
   int e,d;//THE ENCRYPTING AND DECRYPTING EXPONENTS
   cin>> response;

if(response==1)
 {
   cout <<"\n\nDO YOU ALREADY GAVE A KEY? IF YES ENTER '1', ELSE ENTER
```

```
'0' TO GENERATE A NEW KEY.\nCHOICE: ";
     cin>>response;



 if(response==0)
          {
redo:while(response!=1)
  {
    for(int i=0; i<=1; i++)
    {
                                          // p[i]= choosePrime();
    }

if(((p[0]*p[1])<8989) ||(p[0]==p[1])||((p[0]*p[1])>9999))
                          goto redo;
    cout <<"\n\nDO YOU PREFER " <<p[0] <<" AND " <<p[1] <<" FOR YOUR
PRIMES? ENTER '1' IN AGREEMENT, '0' OTHERWISE.\nDECISION: "; cin>>
response;
                  }

  N= p[0]*p[1];
 int phiN=(p[0]-1)*(p[1]-1), E[5], j=0;
  for(int i=3; i<=(phiN-1); i++)
  {
                  if(gcd(i,phiN)==1)
    {
                  E[j]= i;
       j++;
       if(j==4)
                   break;
    }
                   }


 cout <<"\n\nPLEASE SELECT ONE OF THE NUMBERS BELOW FOR YOUR
ENCRYPTING EXPONENT\n";


for(int i=0; i<=3; i++)
  {
                  cout <<"  " <<E[i];
                  }
```

```
 cout  <<"\nENTER e: ";
 cin >>e;

          d= InvF(e,phiN);

   cout <<"\n\n(" <<e <<"," <<N <<") AND (" <<d <<"," <<N <<") ARE YOUR PUBLIC AND PRIVATE KEYS RESPECTIVELY. THE PRIVATE KEY IS HIGHLY CONFIDENTIAL!!!";

  }
  else
  {
    cout <<"\n\nENTER THE VALUE FOR 'e' AND 'n' RESPECTIVELY.\ne= "; cin>>e;

    cout <<"n= "; cin>>N;

  }

cout <<"\n\nENTER '1' TO PROCEED WITH ENCRYPTION, '0' OTGERWISE\nDECISION: "; cin>> response;

  if(response!=1)
  {
    cout <<"\n\nTHANK YOU FOR GENERATING YOUR KEYS....";
                  exit(0);

  }


          char alp[1039999], watch='a';
 unsigned long long count=0;
  cout <<"\n\nKINDLY PROCEED TO ENTER THE MESSAGE TO BE ENCRYPTED. ENTER '@' AT THE END OF YOUR MESSAGE:\n\n" ;

          while(watch!='@')
  {
                  cin>> watch;
     if(watch=='@')
          break;
    alp[count]= watch;
     ++count;
                  }
```

```
                if((count%2)==1)
                {
                        ++count;
                alp[count]= '.';
                }

 unsigned long long i, r, last=(count/2)-1;
 long n[count/2], num[count];

  for(i=count-1, r= 0; i>=1, r<=last; i= i-2, r++)
  {
    char trans1= alp[i-1], trans2= alp[i];
                n[r]= (assignNum(trans1)*100)+assignNum(trans2);


    n[r]= ModExp(n[r],N,e);

    split(n[r], num[i-1], num[i]);

                }

//THE CIPHER TEXT:
 cout <<"\n\nTHE CIPHER TEXT IS:\n\n";
        for(i=0; i<=(count-1) ; i++)
  {
                cout << assignChar(num[i]);

                }

}

else
        {
     cout <<"\nPROCEED TO INPIT THE DETAILS OF YOUR PRIVATE KEY.\nENTER THE VALUE FOR 'd' AND 'n' RESPECTIVELY." <<endl <<"d= ";

 cin>>d;
    cout <<"n= ";
    cin>>N;

   char alp[1039999], watch='a';
 unsigned long long count=0;
   cout <<"\n\nKINDLY PROCEED TO ENTER THE MESSAGE TO BE DECRYPTED. ENTER '@' AT THE END OF YOUR MESSAGE:\n\n" ;
```

```
        while(watch!='@')
  {
                cin>> watch;
    if(watch=='@')
                        break;
   alp[count]= watch;
    ++count;
                }

                if((count%2)==1)
                 ++count;


 unsigned long long i, r, last=(count/2)-1;
 long n[count/2], num[count];

  for(i=count-1, r= 0; i>=1, r<=last; i= i-2, r++)
  {
    char trans1= alp[i-1], trans2= alp[i];
                n[r]= (assignNum(trans1)*100)+assignNum(trans2);


   n[r]= ModExp(n[r],N,d);

   split(n[r], num[i-1], num[i]);

                }

//THE PLAIN TEXT:
 cout <<"\n\nTHE PLAIN TEXT IS:\n\n";
        for(i=0; i<=(count-1) ; i++)
  {
                cout << assignChar(num[i]);

                }

}

  cout <<"\n\nDO YOU WISH TO UNDERGO THE PROCESS AGAIN?\nENTER '1'
FOR YES, '0' OTHERWISE: ";
     cin >>response;
}
while(response !=0);
```

```
   return 0;
}


//####################
int choosePrime()
{

          srand(time(0));

 unsigned long long answer=0, value;


   while(answer != 2)
   {
                                    value= GetRand();
    answer= ModExp(2,value,value);

   }

 return value;
}

//####################
unsigned long long GetRand()
{

return (rand()%991)+9;

}

//####################
unsigned long long ModExp( unsigned long long a, unsigned long long n, unsigned
long long p)
{

 unsigned long long excess=1;

  while(p != 1)
   {
```

```
    if((p%2)==1)
     {

      excess= (excess*a)%n;

            }

     a= (a*a)%n;
     p = p/2;

    if(p==1)
     {
      a=(excess*a)%n;

            }

          }

    ;
return a;

}



//####################
int gcd(int s,int r)
{
        while(s != r)
  {
                if(s>r)
                s -= r;
        else if(r>s)
                r -= s;

        }

                        return s;
}


//####################
int InvF(int q, int r)
{
```

```
   int d, ans;
   for(d=1; d<r; d++)
   {
          ans= (q*d)%r;
      if(ans == 1)
                  break;
   }


         return d;

}

//####################
long assignNum(char alphTrans)
{
  long x;
  switch(alphTrans)
   {
                        case '~': x=0;break;
                        case '!': x=1;break;
                        case '"': x=2;break;
                        case '#': x=3;break;
                        case '$': x=4;break;
                        case '%': x=5;break;
                        case '&': x=6;break;
                        case '\'': x=7;break;
                        case '(': x=8;break;
                        case ')': x=9;break;
                        case '*': x=10;break;
                        case '+': x=11;break;
                        case ',': x=12;break;
                        case '-': x=13;break;
                        case '.': x=14;break;
                        case '/': x=15;break;
                        case '0': x=16;break;
                        case '1': x=17 ;break;
                        case '2': x=18;break;
                        case '3': x=19;break;
                        case '4': x=20;break;
                        case '5': x=21;break;
                        case '6': x=22;break;
                        case '7': x=23;break;
```

```
case '8': x=24;break;
case '9': x=25;break;
case ':': x=26;break;
case ';': x=27;break;
case '<': x=28;break;
case '=': x=29;break;
case '>': x=30;break;
case '?': x=31;break;
case 'A': x=32;break;
case 'B': x=33;break;
case 'C': x=34;break;
case 'D': x=35;break;
case 'E': x=36;break;
case 'F': x=37;break;
case 'G': x=38;break;
case 'H': x=39;break;
case 'I': x=40;break;
case 'J': x=41;break;
case 'K': x=42;break;
case 'L': x=43;break;
case 'M': x=44;break;
case 'N': x=45;break;
case 'O': x=46;break;
case 'P': x=47;break;
case 'Q': x=48;break;
case 'R': x=49;break;
case 'S': x=50;break;
case 'T': x=51;break;
case 'U': x=52;break;
case 'V': x=53;break;
case 'W': x=54;break;
case 'X': x=55;break;
case 'Y': x=56;break;
case 'Z': x=57 ;break;
case '[': x=58;break;
case '\\': x=59;break;
case ']': x=60;break;
case '^': x=61;break;
case '_': x=62;break;
case '`': x=63;break;
case 'a': x=64;break;
case 'b': x=65;break;
case 'c': x=66;break;
case 'd': x=67 ;break;
```

```
        case 'e': x=68 ;break;
        case 'f': x=69 ;break;
        case 'g': x= 70 ;break;
        case 'h': x= 71 ;break;
        case 'i': x= 72 ;break;
        case 'j': x= 73 ;break;
        case 'k': x=74 ;break;
        case 'l': x=75 ;break;
        case 'm': x=76 ;break;
        case 'n': x=77 ;break;
        case 'o': x=78 ;break;
        case 'p': x=79 ;break;
        case 'q': x=80;break;
        case 'r': x=81;break;
        case 's': x=82;break;
        case 't': x=83;break;
        case 'u': x=84;break;
        case 'v': x=85;break;
        case 'w': x=86;break;
        case 'x': x=87 ;break;
        case 'y': x=88;break;
        case 'z': x=89;break;
        case '{': x=90;break;
        case '|': x=91;break;
        case '}': x=92;break;
    //??????????????????????
        case '^': x=93;break;//136
        case '¬': x=94;break;//172
        case '~': x=95;break;//152
        case 'º': x=96;break;//186
        case '²': x=97;break;//178
        case '³': x=98;break;//179
        case '¹': x=99;break;//185
                //default:
                 }

  return x;
}
```

```
//####################
char assignChar(int numTrans)
{
        char x;
 switch(numTrans)
                        {
                        case 0:  x=' ';break;
                        case 1:  x='!';break;
                        case 2:  x='"';break;
                        case 3:  x='#';break;
                        case 4:  x='$';break;
                        case 5:  x='%';break;
                        case 6:  x='&';break;
                        case 7:  x='\'';break;
                        case 8:  x='(';break;
                        case 9:  x=')';break;
                        case 10:  x='*';break;
                        case 11: x='+';break;
                        case 12: x=',';break;
                        case 13: x='-';break;
                        case 14: x='.';break;
                        case 15: x='/';break;
                        case 16: x='0';break;
                        case 17: x='1';break;
                        case 18: x='2';break;
                        case 19: x='3';break;
                        case 20: x='4';break;
                        case 21: x='5';break;
                        case 22: x='6';break;
                        case 23: x='7';break;
                        case 24: x='8';break;
                        case 25: x='9';break;
                        case 26: x=':';break;
                        case 27: x=';';break;
                        case 28: x='<';break;
                        case 29: x='=';break;
                        case 30: x='>';break;
                        case 31: x='?';break;
                        case 32: x='A';break;
                        case 33: x='B';break;
                        case 34: x='C';break;
                        case 35: x='D';break;
                        case 36: x='E';break;
                        case 37: x='F';break;
```

```
case 38: x='G';break;
case 39: x='H';break;
case 40: x='I';break;
case 41: x='J';break;
case 42: x='K';break;
case 43: x='L';break;
case 44: x='M';break;
case 45: x='N';break;
case 46: x='O';break;
case 47: x='P';break;
case 48: x='Q';break;
case 49: x='R';break;
case 50: x='S';break;
case 51: x='T';break;
case 52: x='U';break;
case 53: x='V';break;
case 54: x='W';break;
case 55: x='X';break;
case 56: x='Y';break;
case 57: x='Z';break;
case 58: x='[';break;
case 59: x='\\';break;
case 60: x=']';break;
case 61: x='^';break;
case 62: x='_';break;
case 63: x='`';break;
case 64: x='a';break;
case 65: x='b';break;
case 66: x='c';break;
case 67: x='d';break;
case 68: x='e' ;break;
case 69: x='f' ;break;
case 70: x='g' ;break;
case 71: x='h' ;break;
case 72: x='i' ;break;
case 73: x='j' ;break;
case 74: x='k';break;
case 75: x='l';break;
case 76: x='m';break;
case 77: x='n';break;
case 78: x='o';break;
case 79: x='p';break;
case 80: x='q';break;
case 81: x='r';break;
```

```
                case 82: x='s';break;
                case 83: x='t';break;
                case 84: x='u';break;
                case 85: x='v';break;
                case 86: x='w';break;
                case 87: x='x';break;
                case 88: x='y';break;
                case 89: x='z';break;
                case 90: x='{';break;
                case 91: x='|';break;
                case 92: x='}';break;
                //??????????????????
                case 93: x='^';break;
                case 94: x='¬';break;
                case 95: x='~';break;
                case 96: x='º';break;
                case 97: x='²';break;
                case 98: x='³';break;
                case 99: x='¹';break;
                //default: x='£';break;
        }

        return x;
}



void split(long nnumber, long &a, long &c)
{


//initialisation of counter
        int ncounter=2;

        while(0<=ncounter)
  {//The computation below extracts the digits in two's from the 4 digit variable
"nnumber" one after the other
    if(ncounter==2)
     {
                a= nnumber/pow(10,ncounter);

                nnumber -= pow(10,ncounter)*a;
         }
     else
```

```
    {
        c= nnumber/pow(10,ncounter);

                nnumber -= pow(10,ncounter)*c;
         }
                ncounter -=2;
   }


}
```