The Role Of Prime Numbers in RSA Cryptosystems

Henry Rowland

December 5, 2016

Abstract

Prime numbers play an essential role in the security of many cryptosystems that are currently being implemented. One such cryptosystem, the RSA cryptosystem, is today's most popular public-key cryptosystem used for securing small amounts of sensitive information. The security of the RSA cryptosystem lies in the difficulty of factoring an integer that is the product of two large prime numbers. Several businesses rely on the RSA cryptosystem for making sure that sensitive information does not end up in the wrong hands. This paper highlights the importance of prime numbers in modern day implementations of RSA cryptosystems. We will discuss how an RSA cryptosystem can be successfully implemented, some of the methods used to find primes considered appropriate for RSA cryptosystems, as well as cryptanalysists techniques for factoring a product consisting of such primes.

1 Introduction

Throughout history, there has been a need for secure, or private, communication. In war, it would be devastating for the opposing side to intercept information during communication over an insecure line about the plan of attack. With all of the information that is stored on the internet today, it is important that sensitive information, such as a person's credit card number, is stored securely. Cryptosystems, or ciphers, use a key, or keys, for encrypting and decrypting information being communicated with the intention of keeping this information out of the hands of unwanted recipients. Prior to the 1970s, the key(s) used in cryptosystems had to be agreed upon and kept private between the originator, or sender, of a message and the intended recipient. This made it difficult to achieve secure communication for two parties far from one another because they would have to find another means of secure communication to agree upon the key(s) of the cryptosystem begin used. Such ciphers, where the encryption/decryption key(s) must be kept private between the originator and intended recipient, are called symmetric-key ciphers.

As technology increased, the security of symmetric-key ciphers became more vulnerable. It was not until the 1970s that a more secure cryptosystem, the RSA cryptosystem, was made public by Rivest, Shamir and Adleman, three MIT researchers. The RSA cipher, when implemented appropriately, is still considered to be an unbreakable cipher. It was the first specific type of asymmetric-key cipher, or public-key cipher, meaning that two parties could publicly agree upon encryption keys over an insecure communication line and not compromise the security of the cipher. The security of the RSA cipher comes from the general difficulties of factoring integers that

are the product of two large prime numbers. The level of security for the RSA cipher increases as the size of the prime numbers used for determining the encrpytion key increases.

Modern implementations of the RSA cipher require that the prime numbers for determining the encryption key be very large, hundreds of digits in length. This paper discusses some of the algorithms, called primality tests, that are used to determine whether or not a given positive integer is prime, very likely prime, or composite. These tests are very helpful for determining the primarily of integers that are hundreds of digits long. There is no known factoring method for determining the prime factors in a feasible time frame for an integer that is the product of primes equal in size to those desired in the RSA cipher. However, to truely understand how the RSA cipher gets its security and the role prime numbers play in this security, one must understand the processes for factoring integers with prime factors. We will dicuss some of the classical factoring techniques and explore some of the modern factoring techniques.

2 Essential Theorems and Definitions

The purpose of this section is to provide useful theorems, proofs and definitions for understanding the information presented in this paper. We will refer to the theorems and definitions presented in this section throughout the remainder of this paper.

The Euclidean Algorithm

The Euclidean algorithm is a historically famous mathematical algorithm that was described around 300 BC [2] and provides efficient methods for finding greatest common divisors and multiplicative inverses for modular arithmetic when the modulus is large, as in the case of RSA ciphers. First we will discuss the initial part of the Euclidean algorithm, which can be used for determining greatest common divisors of two positive integers.

Let a and b be positive integers with $a > b$. Suppose we want to determine the greatest common divisor of a and b, or $\gcd(a, b)$. If $b|a$, it follows then that $\gcd(a, b) = b$. Otherwise, if $b \nmid a$, we can find $\gcd(a, b)$ by applying the Euclidean algorithm and using repeated division. We first divide b into a to obtain the quotient q_1 and the nonnegative remainder r_1, where $r_1 < b$. We can now write

$$a = q_1 b + r_1.$$

Next, we divide r_1 into b and obtain the quotient q_2 and positive remainder r_2, where $r_2 < r_1 < b$. Then we write

$$b = q_2 r_1 + r_2.$$

We repeat this process until the remainder equals zero. The following is the general equation resulting from the nth division.

$$r_{n-2} = q_n r_{n-1} + r_n.$$

If the next division yields a 0 remainder ($r_{n-1} = q_{n+1} r_n$), then it follows that $\gcd(a, b) = r_n$.

As mentioned, we can use the Euclidean algorithm for determining multiplicative inverses for modular arithmetic.The following theorem provides the basis for this concept.

Theorem 1. *If a and b are integers with at least one nonzero, then there exist integers s and t such that* $\gcd(a, b) = as + bt$.

Let a and n be positive integers with $\gcd(a, n) = 1$. We wish to find $a^{-1} \pmod{n}$. Since $\gcd(a, n) = 1$, it follows from the above theorem that there exist integers s and t such that,

$$1 = sn + ta.$$

We can then solve the above equation for ta, which yields

$$ta = 1 - sn.$$

Reducing this equation modulo n will give the following.

$$\begin{aligned} ta &\equiv (1 - sn) \pmod{n} \\ &\equiv (1 - 0) \pmod{n} \\ &\equiv 1 \pmod{n} \end{aligned}$$

Hence, $ta \equiv 1 \pmod{n}$ and it follows that the value of $t \pmod{n}$ is equivalent to $a^{-1} \pmod{n}$.

Euler's Theorem

Definition 1. *(Euler's Phi Function) Let n be a positive integer. Then Euler's Phi function for n, denoted $\phi(n)$, is equal to the number of integers a such that $1 \leq a \leq n$ such that* $\gcd(a, n) = 1$.

Theorem 2. *(Fermat's Little Theorem) If p is a prime number and a is an integer with* $\gcd(a, p) = 1$*, then $a^{\phi(p)} \equiv 1 \pmod{p}$.*

A prrof of Fermat's Little Theorem can be found in [3] on page 80.

Theorem 3. *(Euler's Theorem)Let $n \in \mathbb{Z}^+$ and $a \in \mathbb{Z}$. If* $\gcd(a, n) = 1$*, then $a^{\phi(n)} \equiv 1 \pmod{n}$.*

a proof of Euler's Theorem may be found in [3] on page 82.

Lemma 1. *If p is a prime number, then $\phi(p) = p - 1$.*

Proof. Let p be a prime number. Since p is prime, it follows that every positive integer less than p is relatively prime to p. Hence, there are $p-1$ integers that relatively prime to p and $\phi(p) = p-1$. □

Lemma 2. *If p and q are prime integers, then $\phi(pq) = \phi(p)\phi(q)$.*

Proof. Let p and q be distinct primes. Then there are $q - 1$ positive integers that are less than pq and not relatively prime to p and $p - 1$ positive integers that are less than pq and not relatively prime to q. Since p and q are distinct primes, the only integer that is a multiple of both p and q is pq. Hence, there are $1 + (p - 1) + (q - 1)$ integers that are not relatively prime to pq. We can then subtract the total number of integers that are not relatively prime to pq from the product pq to determine $\phi(pq)$, as follows:

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

Thus, $\phi(pq) = \phi(p)\phi(q)$. □

3 Public-Key Ciphers

Prior to the 1970s, all ciphers were symmetric-key ciphers. With symmetric-key ciphers, users agree on encryption and decryption keys that are clearly related, many times identical, and which must both be kept private between the two users [2]. In symmetric-key ciphers, the decryption key can easily be obtained from knowing the the encryption key. The security of these ciphers is dependent upon keeping the keys used for encryption and decryption a secret from any outsiders. For example, suppose two parties were using a symmetric-key cipher to communicate a message privately. If an outsider were happen to gain possesion of the encryption or decryption keys for the cipher, the outsider would be able to break the cipher and read the message.

Before the 1970s, an individual key would have to be securely transported to every intended member of the communication network. Any expansion in the communication network would eventually be overwhelmed with the burden of key distribution [4]. In other words, there is a fundamental problem of sharing secret information for establishing the keys of a symmetric-key cipher [3].

Fortunately there exist ciphers such that two parties are able to publicly agree upon encryption keys over an insecure communication line and not compromise the security of the cipher. Such ciphers are asymmetric ciphers and are commonly called public-key ciphers. In a public-key, or asymmetric, cipher, users have their own set of keys. The encryption key for a user is made public while the decryption key is kept private and is also computationally infeasible to deduce from the encryption key. The concept behind public key ciphers is that it should be easy to encrypt a plaintext message using the public encryption key but very difficult to decrypt the message without certain information about the decryption key. This idea for public-key ciphers was first brought to the public's attention by Whitfield Diffie and Martin E. Hellman in their 1976 paper, *New Directions in Cryptography* [5]. Diffie and Hellman are often credited as the first to originate the idea of public-key ciphers. However, in 1997, Clifford Cocks gave a presentation where he revealed that the concept of public-key ciphers was invented by James Ellis in 1970, a few years before Diffie and Hellman's paper [2].

During the 1970s, Cocks and Ellis worked together for the British government, at the Government Communications Headquarters (GCHQ), where all of their work was kept private by the British government. In 1970, Ellis proved to himself that public-key cryptography was possible but could not provide a specific type of public-key cipher. This was also the case with Diffie and Hellman, when they first revealed the idea of public-key ciphers. Ellis knew that he needed to determine a special one-way function; that is, a function that could be reversed only if the receiver had a special piece of information. After three years of unsuccessfully attempting to find a one-way function that would satisfy Ellis's requirements, Clifford Cocks, a mathematician who specialized in number theory, joined the GCHQ. Six weeks after Cocks joined the GCHQ, a member of the GCHQ mentioned Ellis's idea of public-key and how the GCHQ had been working to find a mathematical function that could make the idea work to Cocks. That very afternoon, Cocks had successfully formulated the first specific type of asymmetric public-key cipher, which is a special version of the cipher that would become known as the RSA cipher [4].

4 The RSA Cipher

The RSA cipher was the first specific type of public-key cipher. It was named after Ron Rivest, Adi Shamir, and Len Adleman, who were three MIT researchers that are credited for being the first to publiclly present the RSA cipher [2]. As we mentioned previously, Cocks formulated a version of this cipher four years before Rivest, Shamir, and Adleman's version was made public. Since its debut, the RSA cipher has become one of the most widely used cryptosystems in the world. For example, the RSA cipher is used to secure our information for situations such as when we purchase anything with a credit card over the Internet or use an ATM [2].

The RSA cipher is an asymmetric cipher based on the theoretical work of Diffie and Hellman in what is known as the Diffie-Hellman key exchange algorithm [7]. While the algorithm for the RSA cipher is relatively new, the basis for the decryption process in the RSA cryptosystem is a 300 year old theorem, called Euler's Theorem. RSA ciphers are implemented using prime numbers and modular arithemetic with exponentation [2]. As we will show in later sections, prime numbers, especially large prime numbers, play a vital role in the implementation of the RSA cipher. Modern implementations of RSA ciphers use primes that are hundreds of digits in length. To this day, there are no known algorithms that are able to determine the prime factors of composite numbers that are several hundreds of digits in length in a practical amount of time. This is the reason why RSA ciphers have such a high level of security [1].

In this section we will explain how the RSA cipher is implemented, why it works, and how it remains secure.

4.1 Numerical Representation of Messages

Before communicating a message with RSA ciphers, the plaintext message is converted into numeric form. In other words, the plaintext message will be represented by one or more positive integers. For example, each letter of the alphabet can be represented by consecutive integers as in the following table.

Table 1: Numerical Representation Of The English Alphabet.

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Using Table 1 above, each letter in the plaintext message **MATH** will be represented in numeric form by the integers $12, 0, 19, 7$ respectively. However, if we represent plaintext messages in this fashion, we have no way to determine capitalization, nor could we represent any of the characters used in the English language for punctuation. There is also no way for us to represent numbers in our message. Hence, it is much more popular to express plaintext messages using ASCII characters. The following figure, provided by the Communications Museum of Macao, shows how each ASCII character is represented numerically [8].

Figure 1: Integer representation of ASCII characters.

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
32	[space]	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	[backspace]

Now, using the table in the above figure, we are able to express each character in the plaintext message **December 29, 1993**, with the integers

$$68, 101, 99, 101, 109, 98, 101, 114, 32, 50, 57, 44, 32, 49, 57, 57, 51,$$

respectively. Note that we would not be able to represent the comma and the numbers of the date in the above message numerically using Table 1.

4.2 Implementing the RSA Cipher for Secure Communication

In this section, we discuss how RSA ciphers are implemented, as well as how they are able to achieve communication. Suppose the originator (the sender) of a plaintext message wants to send the numerical representation of the message to the intended recipient over an insecure communication line using an RSA cipher. The originator would use the basic steps for implementing an RSA cipher detailed in the following paragraphs.

The intended recipient begins the process by generating the public encryption key (e, n). The recipient chooses two distinct prime numbers p and q, and computes $n = pq$ and $\phi(n) = (p-1)(q-1)$, where ϕ denotes Euler's Phi Function. Next, the recipient chooses an integer e, called the encryption exponent, such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$ (**Note:** In theory, we just need e to be a positive integer with $\gcd(e, \phi(n)) = 1$). The recipient then makes values e and n public and keeps the values for p and q a private.

Let each integer of the plaintext message be represented by some integer x, where x is less than n. The originator of the message converts the plaintext message using a prearranged conversion method (such as in Table 1 or Figure 1) to one or more positive integers x, where $x < n$. To encrypt each x in the plaintext message, the originator determines the corresponding ciphertext integer y by calculating

$$y \equiv x^e \pmod{n}.$$

In the case that the plaintext message is too large for one calculation, the originator simply breaks the numeric message into blocks, each of which is less than n and applies the ecryption algorithm to each block.

The originator then sends the resulting ciphertext y values to the intended recipient over the insecure communication line. To decrypt the ciphertext integer(s) y, the recipient first determines the decryption exponent d, where $d \equiv e^{-1} \pmod{\phi(n)}$. Note that d is also kept private. Finally, the recipient decrypts y and converts the corresponding plaintext integer x by calculating

$$x \equiv y^d \pmod{n}.$$

There are several aspects in the above process that discuss below in order for one to grasp a true understanding of how the RSA cipher works. First, we discuss how one chooses primes p and q in the process for generating the encryption key (e, n). The primes p and q should be chosen randomly and be independent of each other. In addition, these primes should also be very large. The size of p and q chosen depends on the level of security desired. The larger p and q are, the more secure the cipher will be. As we will see later, it is also necessary that p and q not be too close to one another. In modern RSA ciphers, the size of the primes desired are hundreds of digits in length. Determining the primality of an integer that is hundreds of digits in length seems like a rather daunting and time-consuming task. However, we will discuss in later sections that that there are algorithms, called primality tests, for determining the primality of such large integers can be done fairly quickly [3].

How does the originator of a message obtain the encryption key from the intended recipient? If the intended recipient has not already generated an encryption key and made it public, the recipient has to be made aware that the originator wants to send the recipient a secure message using an RSA cipher. The recipient selects primes p and q as discussed in the previous paragraph. Next, the recipient determines the values of e and n for the encryption key and send these values (e, n) to the originator over an insecure line of communication. If there are a network of people who frequently communicate messages using an RSA cipher, each person in that network could choose their own distinct values for primes p and q to calculate the encryption key (e, n). The network could then make each person's encryption key (e, n) public in a directory. If the originator wanted to send a message to a member in the network, the originator could simply look up the encryption key (e, n) for that member and encrypt the message. This process would be similar to looking up a person's number in a phone book.

Next we explain why it is desired that the encryption exponent e in the RSA cipher is chosen such that $\gcd(e, \phi(n)) = 1$. Recall that the decryption exponent d is given by
$d \equiv e^{-1} \pmod{\phi(n)}$. Choosing an e such that $\gcd(e, \phi(n)) = 1$ guarantees the existence of the multiplicative inverse $e^{-1} \pmod{\phi(n)}$, which is the decryption exponent d [2].

Finally, we discuss why the original message x can be recovered by calculating $y^d \pmod{n}$; that is, why $x \equiv y^d \pmod{n}$. Assume that the intended recipient of a message that was encrypted using an RSA cipher chose p and q to be very large distinct primes. Then it is almost certain that neither p nor q is a factor of x; that is, $\gcd(x, n) = 1$. To see that $\gcd(x, n) \neq 1$ is highly unlikely, we will calculate the probability of discovering a message x that is not relatively prime to $n = pq$

Note that there are a total of n integers from 1 to n. Since $n = pq$, where p and q are distinct primes, it follows that the integers that are not relatively prime to n are

$$p, 2p, 3p, \ldots, qp$$

$$q, 2q, 3q, \ldots, (p-1)q$$

Thus there are $q + p - 1$ possible integers up to n that are not relatively prime to n. Hence the probability that x is not relatively prime to n is given by

$$\frac{q+p-1}{n} = \frac{q}{n} + \frac{p}{n} - \frac{1}{n} = \frac{q}{pq} + \frac{p}{pq} - \frac{1}{pq} = \frac{1}{p} + \frac{1}{q} - \frac{1}{pq}.$$

Hence, if p and q are both larger than 10^{100}, then this probability is less than 10^{-99}.

From the problem above, we can see that it is safe to assume $\gcd(x, n) = 1$. Further, since d is the multiplicative inverse of e modulo $\phi(n)$, we have $de \equiv 1 \pmod{\phi(n)}$. Using the definition of congruence, it follows that $de = 1 + k\phi(n)$ for some integer k. Using Euler's Theorem and recalling that $g \equiv x^e \pmod{n}$, observe the following:

$$y^d \equiv (x^e)^d \equiv x^{1+k\phi(n)} \equiv x(x^{\phi(n)})^k \equiv x(1^k) \equiv x \pmod{n}.$$

Thus, $x \equiv y^d \pmod{n}$.

In the real world, the RSA cipher is practically never used to encrypt messages more than a few words. The reason is because there is so much computation involved with RSA cipher, making it a slow process. Because of this, RSA cryptosystems (and public-key cryptosystems in general) is more often used as a solution to the key-management problem, rather than actually communicating full messages. In other words, the RSA cipher is used to distribute private keys, which are used to encrypt and decrypt messages with a symmetric cipher [16].

5 The Security of RSA Ciphers: Integer Factorization

It is important to understand how the RSA cipher remains secure after making the values for e and n in the encryption key public. Recall that for public-key ciphers a special function is needed that can only be reversed if the receiver has a special bit of information. In the RSA cipher, this special bit of information are the values of p and q. In general, if an outsider wanted to attack an RSA cipher, the cryptanalyst would have to find the value of the decryption exponent d. We can assume that the outsider would know the values for e and n, since these values are made public. However, if the primes p and q are large, knowing the public key (e, n) would not be enough to determine d. In order for the outsider to determine d, one would have to factor n to find p and q so that $\phi(n)$ could be determined, something that would be practically impossible if p and q are both hundreds of digits long. The key to achieving a high level of security for RSA ciphers is to choose p and q to be very large. The larger p and q are, the more difficult it becomes to factor n; that is, the RSA cipher's high level of security is due to the difficulty of finding p and q from factoring n [2]. In this section we will discuss methods for integer factorization and explain why it is so difficult to determine the prime factors p and q from n, if p and q are of size ideal for use in RSA ciphers.

5.1 Integer Factorization

To break an RSA cipher, the only real difficulty an attacker will face is in factoring $n = pq$ to find p and q. Since it is believed to be a very difficult task, the factorization of integers composed of two distinct, very large prime factors forms the basis of most modern cryptosystems. The development

of public-key ciphers in the 1970s, such as the RSA cryptosystem, has amplified research in the area of integer factorization. Factoring a composite integer into prime factors, such as p and q in an RSA cipher, is one of the greatest problems in mathematics. The most trivial technique for finding the prime factorization of a positive integer is to simpily perform division. For example, we could determine the prime factors of n by checking to see if each prime number $q \leq \sqrt{n}$ divides n. The problem with this approach for finding the prime factorization of n is that if n were composed of only extremely large prime factors, as desired for RSA ciphers, there would potentially be a tremendous amount of primes to test before finding one that divides n. To date, there is no factoring algorithm that has proven to be efficient enough to determine the prime factors of integer composed of primes that are hundred of digits in length. In this section, we will discuss Fermat Factorization, a classical integer factorization, and Shor's Factoring algorithm, the most efficient factoring algorithm known today [6].

5.1.1 Fermat Factorization

Fermat factorization is a technique for determining the distinct prime factors of an integer that are relatively close together. We will see that the further apart these prime factors are from one another, the less efficient this algorithm becomes. We will explain why this is the case after first explaining how to implement Fermat factorization.

Suppose for a given positive integer n, where $n = pq$ and p and q are two distinct prime integers, we wish to determine p and q. Let $x = \frac{p+q}{2}$ and $y = \frac{p-q}{2}$. Then $x^2 - y^2$ can be simplified as follows.

$$\begin{aligned} x^2 - y^2 &= (x+y)(x-y) \\ &= (\frac{p+q}{2} + \frac{p-q}{2})(\frac{p+q}{2} - \frac{p-q}{2}) \\ &= (\frac{2p}{2})(\frac{2q}{2}) \\ &= pq \end{aligned}$$

Thus,

$$n = pq = x^2 - y^2 = (x+y)(x-y).$$

It follows that p and q must be $(x+y)$ and $(x-y)$. Hence, to determine p and q, we only need to find x and y, which we describe below.

Because of the Fermat factorization, when successfully implementing the RSA cipher, the intended recipient not only needs to select relatively large prime factors p and q, but $|p-q|$ needs to be large. Note that we can rewrite $n = x^2 - y^2$ as $y^2 = x^2 - n$. In order to determine x and y, we first find the smallest integer k for which $k^2 \geq n$. Next, we calculate the following sequence of integers until we obtain a perfect square:

$$k^2 - n, (k+1)^2 - n, (k+2)^2 - n, \ldots.$$

We know that we will eventually acquire a perfect square since $(\frac{n+1}{2})^2 - n = (n - \frac{1}{2})^2$, which results in the trivial factorization. Further, note that only when p and q are relatively close together, the number of choices for x required in Fermat factorization process is relatively small. For finding p and q from $n = pq$, Fermat factorization requires an average of $\frac{|p-q|}{2}$ steps; that is, the number of steps required in Fermat factorization for finding p and q for $n = pq$ is dependent on the distance between p and q.

5.1.2 Quantum Computing and Shor's Factoring Algorithm

In 1994, Peter Shor presented an algorithm that, if used on a quantum computer, is able to factor large integers in polynomial time. This algorithm is exponentially faster than any classical factoring algorithm. Shor's factoring algorithm has since become the most famous application of quantum computers [6]. In this section we will explain how a quantum computer works as well as brief on the implementation of Shor's factoring algorithm.

Quantum computation is a computing paradigm where data is encoded in the state of objects that are governed by the laws of quantum physics. Quantum computation can only be performed on quantum computers, which at this time are generally too expensive to consider commercial availability [14].

Quantum computation involves the use of quantum mechanics, which are a completely new model for computation and may eventually provide ways to factor large integers in a practical amount of time, hence breaking the RSA cryptosystem. Typical household computers represent information in terms of bits (1's or 0's) and perform computations using logical operations (i.e. NOT, AND, OR) on these bits. Rather than being in a fixed state, such as 0 and 1, a quantum system may be in linear combination of states, called a super position of states. Information in quantum computers is represented by quantum bits, or qubits, each of which is a linear combination of 0 and 1. The idea behind quantum computing is that a quantum computer at it logical level will act according to the principles of quantum mechanics [13].

In order to implement Shor's factoring algorithm effectively, the algorithm should be implemented on a quantum computer. The steps for Shor's algorithm involve the use of quantum Fourier transforms and storing data into registers on a quantum computer, which are designated spaces of memory for storing data in qubits [15]. Further information regarding the steps of Shor's quantum factoring algorithm can be found in **An Analysis of Quantum Factoring Using Shors Algorithm** [11].

5.2 The RSA Factoring Challenges

The founders of the RSA cryptosystem understood that it was the general difficulty of factoring an integer n that is the product of two large distinct prime numbers. Thus, they knew that research in integer factorization would be important to the future of the RSA cryptosystem. As an attempt to motivated research in integer factorization, shortly after introducing the RSA cryptosystem in 1977, Rivest, Shamir, and Adleman proposed the challenge of breaking a specific RSA cipher to the public. The modulus ($n = pq$) for this cipher was 129 digits in length and was known as RSA-129 [10]. Anyone able to successfully break this cipher by factoring this n within five years would be rewarded with $100. Given the technology available and factoring techniques known at the time this challenge was first proposed, it was estimated that it would take 40 quadrillion years to factor RSA-129. A team of 600 researchers from around the globe were finally able to factor RSA-129 in 1994 with the use of 1600 computers. The whole process took the team a little over six months [2].

To generate greater motivation for research in integer factorization, RSA laboratories presented a broader challenge in 1991 by publishing a list of integers, each of which was the product of two large primes, and offering larger cash prizes for the prime factors. The RSA Factoring Challenge was later deemed inactive in 2007, however, prior to that time, the largest of the posted integers to

be successfully factored was 193 digits long and is shown below.

$$3107418240490043721350750035888567930037346022842727545720161948823206440518081504556346829671723286782437916272838033415471073108501919548529007337724822783525742386454014691736602477652346609$$

It was not until 2005, 14 years after the list of challenge integers was posted, that this 193 digit integer was factored. The process for factoring the above integer took a team of researchers a little over five months and earned the team $20,000 [13].

6 Primality Testing

Since the development of public-key cryptography, interest of finding large prime numbers and studying the characteristics of primes has dramatically increased [7]. In 1971, around the time the idea of public-key cryptosystems was being developed, the largest known prime was 6,002 digits long. A little more than a decade later in 1983, the record prime contained 39,751 digits. As of September 2016, the largest known prime number consists of 22,338,618 digits [9]. In modern implementations of the RSA cipher, it is generally required that the prime numbers p and q chosen to generate the encryption key should be at least a couple hundred digits long to ensure the security of the cipher. However, determining whether a given integer that is hundreds of digits long is prime or composite is an age-old problem in mathematics [1].

Prime numbers are scattered randomly throughout the number line of positive integers. In other words, there is no pattern of prime numbers among composites. If we were to search the set of positive integers for prime numbers, we would see that at times there can be arbitrarily large gaps of composites in between two prime numbers and other times we would run across twin primes, which are primes that are consecutive odd integers (ex: $(3, 5)$, $(5, 7)$, $\ldots$). The *Prime Number Theorem* gives an approximation of how many primes are less than a specific large positive real number. The Prime Number Theorem predicts how many primes numbers are less than a given positive real number, x. For each positive real number x, we let the function $\pi(x)$ represent the number of primes that do not exceed x. The Prime Number Theorem is stated as follows.

Theorem 4. ***(Prime number Theorem)*** *Let x be a positive real number. Let $\pi(x)$ be the number of primes less than x. Then*

$$\pi(x) \approx \frac{x}{\ln x}.$$

In other words, for a random integer selected between 0 and a large integer n, the probability that the selected integer is prime is approximately $\frac{1}{\ln n}$. Hence, as n increases, our chances of randomly selecting a prime number between 0 and n decreases. For example, let $x_1 = 10^{25}$, a number that is 26 digits long, and $x_2 = 10^{50}$, a number that is 51 digits long. Then, the probability that a randomly selected integer between 0 and x_1 is $\frac{1}{\ln(10^{25})} \approx .0174$ and the probability that a randomly selected integer between 0 and x_2 is $\frac{1}{\ln(10^{50})} \approx .0087$. As illustrated from above, the odds of choosing a prime from the range of 0 to a number that is hundreds of digits long are not very good. Let $x_1 = 10^{100}$, a number that is 101 digits long and $x_2 = 10^{200}$, a number that is 201 digits long. Then

the probability that a randomly selected integer between 0 and x_1 is prime is approximately $.0043$ and the probability that a randomly selected integer between 0 and x_2 is prime is approximately $.0022$. The Prime number does not determine the primality of a number, only how likely it is to randomly select a positive real number that is prime. There do, however, exist methods for actually determining the primality of a given positive real number. Such methods are called primality test. A primality test is an algorithm that is used to test whether or not a given positive integer is prime, very likely prime, or composite.

One of the earliest methods for determining whether a given integer is prime or composite is the Sieve of Eratosthenes, which finds all prime numbers less than or equal to a positive integer n. The Sieve of Eratosthenes method for determining the primality of a number dates back to 240BC [9]. The idea behind this method is if n is composite then n has a prime factor less than or equal to $\sqrt{n}$. To implement this method, list all integers from 2 to n. Let x_i be an integer between 2 and n. Then, begining with $x_1 = 2$ cross out all multiples of 2 from the list. Repeat this for each x_i that has not been crossed off of the list. At the end of this method the numbers that remain are the prime numbers up to n [1].

Example 1. *Let $n = 30$. We begin by listing the integers between 2 and 30.*

$$2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30$$

Now we cross off all multiples of 2.

$$2, 3, \cancel{4}, 5, \cancel{6}, 7, \cancel{8}, 9, \cancel{10}, 11, \cancel{12}, 13, \cancel{14}, 15, \cancel{16}, 17, \cancel{18}, 19, \cancel{20}, 21, \cancel{22}, 23, \cancel{24}, 25, \cancel{26}, 27, \cancel{28}, 29, \cancel{30}$$

We have now shortened our list to the following integers:

$$2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29$$

Next we cross off all multiples of 3.

$$2, 3, 5, 7, \cancel{9}, 11, 13, \cancel{15}, 17, 19, \cancel{21}, 23, 25, \cancel{27}, 29.$$

The only integer left in our list that is a multiple of 5 is 25. After crossing 25 off of our list the remaining integers are

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29.$$

Since no integer left in our list is a multiple of another, it follows that the remaining integers are a list of all the prime numbers between 2 and n.

This is an efficient method for determining small primes, but this method is inefficient for determining primes that are hundreds of digits in length, as desired for RSA ciphers. In this section, we discuss more efficient primality tests than the Sieve of Eratosthenes.

There are two types of primality tests, probabilistic primality test and deterministic primality test. A probabilistic primality test is a test or algorithm that determines how likely a given integer is prime while deterministic primality test determines with certainty if a given integer is prime [17]. We will discuss two probabilistic primality test, the Miller-Rabin test and Fermat test, as well as two deterministic test, Lucas's converse of Fermat's Theorem and the AKS algorithm. [1].

6.1 Probabilistic Primality Testing

Since a probabilistic primality test is a test that determines how likely it is that a given integer is prime, the test does not determine for certain that a given integer is prime like a deterministic primality test. However, probabilistic primality tests are in general more efficient algorithms than deterministic primality tests because the exactness in deterministic primality tests require more calculations . In other words, probabilistic primality tests can be implemented at a faster running time. This is why people who work with primality testing still prefer the approaches of probabilistic primality testing even though they might be sacrificing absolute certainty. Contrary to the name, probabilistic primality test really test whether or not a given positive integer is composite. The foundation for many probabilistic primality tests is Fermat's Little Theorem. We will discuss the Fermat test and the Miller-Rabin test.

6.1.1 The Fermat Test

Recall that Fermat's Little Theorem states that if p is a prime number and a is an integer with $\gcd(a, p) = 1$, then $a^{p-1} \equiv 1 \pmod{p}$. The Fermat test uses Fermat's Little Theorem to determine if a given integer is composite. If a given positive integer does not satisfy Fermat's Little Theorem, the Fermat test concludes that the integer is definitely not prime. On the other hand, if the given positive integer does satisfy Fermat's Little Theorem, the Fermat test will conclude that the integer is likely to be prime. We know from Fermat's Little Theorem that if p is prime and a is an integer such that $\gcd(a, p) = 1$, it follows that $a^{p-1`} \equiv 1 \pmod{p}$. Using the contrapositive of Fermat's Little Theorem, a positive integer n is composite if there exists a positive integer a, such that $\gcd(a, n) = 1$ and $a^{n-1} \not\equiv 1 \pmod{n}$. This is the idea behind the Fermat test. We explain how to implement this test in the following paragraph.

Suppose we are trying to decide if the positive integer n is prime or composite. We would begin by calculating $a^{n-1} \pmod{n}$ for each integer a where $a \in \{1, 2, 3 \ldots, n-1\}$. If there is an a such that $a^{n-1} \not\equiv 1 \pmod{n}$, we can conclude with certainty that n is not prime. Otherwise, if $a^{n-1} \equiv 1 \pmod{n}$ for all values of a, we conclude that n is likely to be a prime number, which we refer to as a probable prime. For example, suppose that we want to use Fermat's test to determine if 33 is prime. Note that for $a = 2$, using the method of repeated squaring, $2^{32} \equiv 4 \pmod{33}$. It follows that 33 is definitely not prime since $2^{32} \not\equiv 1 \pmod{33}$.

The Fermat test is relatively easy to implement and determines for certain if a given positive integer is not prime. However, there do exists integer that pass the Fermat test for primality that are not prime. In other words, there are integers n and a for which $a^{n-1} \equiv 1 \pmod{n}$ even though $\gcd(n, a) = 1$ and n is not prime. In this case, n is called a pseudoprime to the base a. There are only 245 pseudoprimes to the base 2 less than one million, while there are a total of 78,498 primes less than one million [2]. This means, given that a positive integer n less than one million passes the Fermat test with $a = 2$, the probability that n is a pseudoprime is $\frac{245}{78743} \approx .003$. There also exists positive integers n that are pseudoprime to every base a from 1 to $n - 1$ with $\gcd(n, a) = 1$ that are not prime. Such numbers are called Carmichael numbers, or absolute pseudoprimes. There are only 2163 Carmichael numbers less than 25 billion. The smallest Carmichael number is 561 [2].

Although the algorithm used in the Fermat test is efficient and easy to perform, it is not a preferred primality test since pseudoprimes and Carmichael numbers satisfy the properties of Fermat's Little Theorem for every base a [1]. However, there are stronger probabilistic primality tests based

on this algorithm that have a lower probability of determining a composite number as a probable prime while also remaining to be efficient. One such primality test is the Miller-Rabin test, which we discuss next.

6.1.2 The Miller-Rabin Primality Test

One of the most popular probabilistic primality test is the Miller-Rabin test, dating back to the 1970s [12]. Like the Fermat test, the Miller-Rabin test does not prove the primality of a given positive integer, but can prove that that integer is not prime. This test is a combination the Fermat test and the basic principle that modular exponentation is accomplished by successive squaring. The Miller-Rabin test has a significantly lower probability of declaring a composite number as a probable prime. For a given positive integer n, the probability that the Miller-Rabin test fails to recognize n as a composite number for a randomly choosen positive integer a less than n is at most $\frac{1}{4}$. If the test is repeated k times, the probability of failure is at most $(\frac{1}{4})^k$. In the range of 1 to 10^{10}, calculating $a^{n-1} \pmod{n}$ in the Fermat test will fail to recognize a composite number less than 1 out of $30,000$ times, while using the Miller-Rabin test with $a = 2$ will fail to recognize a composite number 1 out of $100,000$ times [2]. Hence, there is a greater probability that a given positive integer n is prime if n passes the Miller-Rabin test than if n passes the Fermat test [3]. Next, we discuss the basic steps for implementing the Miller-Rabin test for a given positive integer.

Suppose we want to determine if a positive integer n is composite or likely prime using the Miller-Rabin test. We begin by writing

$$n - 1 = 2^k m,$$

where k and m are positive integers and m is odd. We then choose a random positive integer a such that $1 < a < n - 1$ and compute

$$b_0 \equiv a^m \pmod{n}.$$

If $b_0 \equiv 1 \pmod{n}$ or $b_0 \equiv -1 \pmod{n}$, we stop testing and declare that n is probably prime. Otherwise, we continue testing and compute

$$b_1 \equiv b_0^2 \pmod{n}.$$

Now, if $b_1 \equiv 1 \pmod{n}$, we conclude that n is composite, and if $b_1 \equiv -1 \pmod{n}$, we conclude that n is probably prime and stop testing. Otherwise, we continue testing and compute

$$b_2 \equiv b_1^2 \pmod{n}.$$

We then determine to stop or keep testing from b_2 similar to how we did for b_1. We continue this process until stopping or reaching b_{k-1}. If $b_{k-1} \not\equiv \pm 1 \pmod{n}$, it follows that n is composite. See the following example.

We will illustrate the Miller-Rabin primality test with $n = 113$ and $a = 2$. Let $n = 113$ and $a = 2$. Then $n - 1 = 112 = 2^4(7)$, and we have $k = 4$ and $m = 7$. We begin by computing $b_0 \equiv a^m \pmod{n}$, as follows.

$$\begin{aligned} b_0 &\equiv a^m \pmod{n} \\ &\equiv 2^7 \pmod{113} \\ &\equiv 15 \pmod{113} \end{aligned}$$

Hence $b_0 \equiv 15 \pmod{n}$ and since $b_0 \not\equiv \pm 1 \pmod{113}$, so we continue testing. Now, we calculate $b_1 \equiv b_0^2 \pmod{113}$ as follows.

$$\begin{aligned} b_1 &\equiv b_0^2 \pmod{113} \\ &\equiv 15^2 \pmod{113} \\ &\equiv 112 \pmod{113} \\ &\equiv -1 \pmod{113} \end{aligned}$$

Since $b_1 \equiv -1 \pmod{n}$, we stop testing and declare that $n = 113$ is probably prime.

Suppose we wish to find a prime of about 80 digits. The Prime Number Theorem states that the density of primes around x is approximately $\frac{1}{\ln(x)}$. When $x = 10^{80}$, the Prime Number Theorem gives a density of $\frac{1}{\ln(10^{80})} \approx \frac{1}{184}$. We can disregard the even numbers, and this density can be raised to $\frac{1}{92}$. Next, we pick a random positive integer between 1 and x and throw out all even numbers and the multiples of small primes. We test each remaining integer in succession by the Miller-Rabin test. Doing so tends to eliminate all of the composites. On average, it will take less than 100 repetitions of the Miller-Rabin test to find a likely candidate for a prime [3]. Thus, this test can generally be executed fairly quickly.

6.2 Deterministic Primality Testing

Deterministic primality tests are designed to determine whether a given positive integer is prime or not with certainty. If a given integer passes a deterministic primality test, that integer is certainly prime. The exactness involved in deterministic primality test consequently cause the algorithms used to be more complicated than those used for probabilistic primality test. The most efficient deterministic primality test was discovered in 2002 and uses the AKS algorithm, which is the only known primality test that falls into the category of an unconditional deterministic polynomial-time algorithm' [1]. However, even the AKS primality test is not practical for testing integers of the size desired for RSA ciphers. A study done about the AKS reported that it took roughly seventy minutes to determine that a 25 digit integer was prime using the AKS primality test. Another deterministic primality test uses Lucas's Converse of Fermat's Theorem which places further restrictions of the base a in the congruence $a^{n-1} \equiv 1 \pmod{n}$ from Fermat's Little Theorem. This section discusses both the primality test using Lucas's Converse of Fermat's test and the AKS algorithm.

6.2.1 Lucas's Converse of Fermat's Theorem

Recall from the probabilistic primality test section, the Fermat test uses Fermat's Little theorem to determine if a given positive integer n is likely prime. If n satisfies the conditions in Fermat's Little Theorem, the Fermat test declares that n is likely prime, not definitely prime. A French number theorist, Edouard Lucas, imposed further restrictions on the base a in the congruence $a^{n-1} \equiv 1 \pmod{n}$ from Fermat's Little Theorem to guarantee the primality of n. By doing so, in 1876, Lucas was actually the first person to devise an efficient primality test.

Theorem 5. *Let $n \in \mathbb{Z}^+$ with $n > 1$. If there exists an integer a such that $a^{n-1} \equiv 1 \pmod{n}$ and $a^{\frac{n-1}{p}} \not\equiv 1 \pmod{n}$ for all primes p dividing $n - 1$, then n is prime.*

The above theorem is known as Lucas's Converse of Fermat's Theorem, which was first given in 1876 [10]. Lucas's Converse of Fermat's Theorem can be used to determine with certainty whether or not a given positive integer is prime.

For example, we will use this theorem to determine the primality of 97 for the base $a = 3$. We have $3^{96} \equiv 1 \pmod{97}$. T Note that the only prime numbers that divide 96 are 2 and 3. We then compute

$$3^{\frac{96}{2}} \equiv 3^{48} \equiv -1 \pmod{97}$$
$$3^{\frac{96}{3}} \equiv 3^{32} \equiv 35 \pmod{97}.$$

Hence, taking Lucas's Converse of Fermat's Theorem into account, 97 must be prime.

Lucas's Converse of Fermat's Theorem was later improved in the late 1960s.

Theorem 6. *Let $n \in \mathbb{Z}^+$ with $n > 1$. If for each prime p_i dividing $n - 1$ there exists an integer a_i such that*
$a_i^{\frac{n-1}{p_i}} \not\equiv 1 \pmod{n}$*, then n is prime.*

Again, we will consider whether or not 97 is prime but we will use this improved version of Lucas's Converse of Fermat's Theorem. Recall that $3^{96} \equiv 1 \pmod{97}$ and the prime divisors of $n - 1 = 96$ are 2 and 3. Now we see for the different bases 3 and 7 that

$$3^{\frac{96}{2}} \equiv 3^{48} \equiv -1 \pmod{97}$$
$$7^{\frac{96}{3}} \equiv 7^{32} \equiv 35 \pmod{97}$$

Thus, we conclude that 97 is a prime number.

The above two theorems may seem relatively simple to implement for determining with certainty whether a given positive integer n is prime or not, however, there can be rather severe difficulties in both methods. The two theorems above only reduce the problem of proving the primality of n to finding the complete factorization of $n - 1$. In many cases, such as determining the primality of integers as large as the primes desired for RSA ciphers, it is no easier of a task to factor $n - 1$ than to factor n. Further, if we were to investigate the primality of a positive integer n that is hundreds of digits in length, we would have to find all prime factors of $n - 1$. This would mean potentially testing the primality of integers less than $n - 1$ and then determining if those primes are a factor of $n - 1$. Thus, this deterministic primality test is computationally infeasible to implement for possible primes that are of desired length for the RSA cipher.

6.2.2 The AKS Algorithm

The AKS primality test is the most efficient deterministic primality test known to date. Although the AKS primality test is not quite efficient enough for practical use on integers hundreds of digits long, it is an important algorithm to study and hopefully improve. In this section we will explain how the AKS is implemented.

The AKS algorithm was considered to be a major breakthrough in 2002 and was proposed by Agrawl, Kayal and Saxena in their paper "Primes is in P" [1]. At the time of their discovery, Kayal and Saxena were undergraduate students, who completed their undergraduate thesis in 2002 under Agrawal's direction. The AKS algorithm was the first deterministic polynoimal-time algorithm for primality testing. It is based on the polynomial generalization of Fermat's Little Theorem given below.

Theorem 7. *Let $a \in \mathbb{Z}_n$ and let $n \in \mathbb{N}$. Then n is prime if and only if*

$$(x+a)^n \equiv x^n + a \pmod{n}.$$

In other words, an integer $n > 1$ is prime if and only if the polynomials $(x+a)^n$ and $x^n + a$ are congruent modulo n [12]. The binomial theorem for expanding $(x+a)^n$ is one of the main ideas behind the AKS test. The coefficients of the polynomial resulting from the expansion of $(x+a)^n$ come from the nth row of Pascal's triangle, shown below.

						1									**row 0**
					1		1								**row 1**
				1		2		1							**row 2**
			1		3		3		1						**row 3**
		1		4		6		4		1					**row 4**
	1		5		10		10		5		1				**row 5**
1		6		15		20		15		6		1			**row 6**
1		7		21		35		35		21		7		1	**row 7**
⋮		⋮		⋮		⋮		⋮		⋮		⋮		⋮	⋮

There is an important observation to note about the difference between the prime rows and the composite rows of Pascal's triangle. Notice that every number, except for the two 1's one the ends of the row, in row 7 of the pascal triangle is divisible by 7. However, 6 does not divide every number in row 6 of Pascal's triangle. The statement in the following theorem holds in general.

Theorem 8. *The integer $n > 1$ is prime if and only if all the numbers in row n (except the 1's at the ends of the row) are divisible by n.*

This observation of the prime rows in Pascal's triangle is particularly interesting when considering the congruence of the polynomials modulo n. Using the binomial theorem, if we raise $(x+1)$ to the 7th power, we get a polynomial whose coefficients are the same numbers in ro 7 of Pascal's triangle. Observe the following.

$$(x+1)^7 = x^7 + 7x^6 + 21x^5 + 35x^4 + 35x^3 + 21x^2 + 7x + 1$$

Note that every coefficient between x^7 and 1 in the above polynomial (7, 21, and 35) are congruent to $0 \pmod{7}$. Hence we can write

$$\begin{aligned}(x+1)^7 &\equiv x^7 + 0x^6 + 0x^5 + 0x^4 + 0x^3 + 0x^2 + 0x + 1 \pmod{7} \\ &\equiv x^7 + 1 \pmod{7}\end{aligned}$$

We can then use this information to test the primality of a given positive integer n by checking that each coefficient of the polynomial that results from expanding $(x+1)^n$ is divisible by n. However, as the size of n increases, so do the number of coefficients to test. This test becomes inefficient when dealing with a positive integer n that is composed of primes that are of the desired size for an RSA cipher. Even if n has only 30 digits or so, it would take eons to compute the polynomial for $(x+1)^n$ [13].

Agrawl, Kayal, and Saxena thought that in order to make this algorithm more efficient, we might consider a polynomial $x^r - 1$ of a sufficiently small degree r and test

$(x-a)^n \equiv x^n - a \pmod{n, x^r - 1}$. That is, when we are dealing with the remainders of $(x-a)^n$ and $x^n - a$ and dividing by $x^r - 1$. We will elaborate further on determining an appropriate r while explaining on the basic steps involved for implementing the AKS primality test.

The AKS algorithm is essentially a combination of three separate algorithms. There is an algorithm for each of the three basic steps for implementing the AKS algorithm. They are the following:

1. Test for perfect powers
2. Find an appropriate r
3. Check $(x-a)^n \equiv x^n - a \pmod{n, x^r - 1}$ for all positive $a \leq 2\sqrt{r}\log n$.

We shall now explain how each of these steps is implemented to perform the AKS primality test for a given positive integer n.

The first step tests for any perfect powers of n. In other words, the first step is used to check if there exists an integer a such that $a^b = n$ for $a \in \mathbb{N}$ and $b \in [2, \log_2 n]$. It follows that n is not a prime if it can be written in the form a^b. If n can be expressed by a^b, the algorithm will output COMPOSITE. The algorithm for this step is given below.

Algorithm for determining if n is a perpect power.

- for $b = 2$ to $\lfloor \log_2 n \rfloor$
  - set $y = \frac{\log n}{b}$
  - set $a = 2^y$
  - if $a^b = n$ output COMPOSITE
  - $b = b + 1$

[1]

Once we have have determined that n is not a perfect power, we move on the next step, finding the appropriate r. The value of r is crucially important in terms of maximum computational time for the AKS algorithm. We want to find an integer r such that the multiplicative order $n \pmod{r}$, denoted $O_r(n)$, is greater than $(\log n)^2$ Consider the following lemma.

Lemma 3. *There exists an $r \in (0, (\log n)^5]$ such that $O_r(n) > (\log n)^2$, where $n > 2$.*

This lemma states that there exists an r such that $O_r(n) > (\log n)^2$ in the range $(0, (\log n)^5]$. To find the appropriate r, we test for all $r \in [2, (\log n)^5]$ to determine if there is a $k \in [1, (\log n)^2]$ such that $n^k \equiv 1 \pmod{r}$. If there is no such k, then we have an appropriate r. The algorithm for finding such an r is given below.

Algorithm for finding the appropriate r.

- Set $r = 2$
- For $r = 2$ to $(\log^5 n) + 1$
  - check $n^k \not\equiv 1 \pmod{r}$ for all k from 1 to $(\log n)^2$

- if $n^k \equiv 1 \pmod{r}$ for any such k then $r = r + 1$
- if $n^k \not\equiv 1 \pmod{r}$ for any such k return r

Once r has been determined, if there exists an $a < r$ such that $\gcd(a, n) \neq 1$, the algorithm will output COMPOSITE. If $\gcd(a, n) = 1$ for $a < r$ and $r < n$, we move on to the third step. In this step we check that for all a such that $1 \leq a \leq \sqrt{\phi(n)} \log n$, the congruence $(x-a)^n \equiv x^n - a \pmod{n, x^r - 1}$ holds. If there does exist an a such that $1 \leq a \leq \sqrt{\phi(n)} \log n$, output COMPOSITE. Otherwise, output PRIME [1].

The full AKS algorithm with the three basic steps combined is given below.

The AKS algorithm.

- for $b = 2$ to $\lfloor \log_2 n \rfloor$
  - set $y = \frac{\log n}{b}$
  - set $a = 2^y$
  - if $a^b = n$ output COMPOSITE
  - $b+ = 1$
- set $r = 2$
- for $r = 2$ to $((\log n)^5) + 1$
  - check $n^k \not\equiv 1 \pmod{r}$ for all k from 1 to $(\log n)^2$
  - if $n^k \equiv 1 \pmod{r}$ for any such k then $r+ = 1$
  - if $n^k \not\equiv 1 \pmod{r}$ for any such k return r
- check if there exists an $a < r$ such that $\gcd(a, n) \neq 1$, if so output COMPOSITE.
- If $\gcd(a, n) = 1$ for $a < r$ and $r < n$
  - check that for all a such that $1 \leq a \leq \sqrt{\phi(n)} \log n$, the congruence $(x - a)^n \equiv x^n - a \pmod{n, x^r - 1}$ holds.
  - if the congruence $(x-a)^n \equiv x^n - a \pmod{n, x^r - 1}$ does not hold, output COMPOSITE
  - if the congruence $(x - a)^n \equiv x^n - a \pmod{n, x^r - 1}$ does hold, output PRIME.

7 Conclusion

The prime numbers that are chosen for p in q in an RSA cipher determine the level of security for that cipher. Given that p and q are not close to one another, the security of an RSA cipher increases as the size of p and q increase. To this day, the RSA cipher is still considered to be an unbreakable cipher and a secure form of communication when the proper precautions, mentioned in this paper, are taken during implementation. However, advances in technology, specifically quantum computing, and further research into integer factorization techniques, such as Shor's Factoring Algorithm, could pose a serious threat to the security of the RSA cryptosystem in the near future. Even so, further research in both primality testing and integer factorization will determine the use of RSA cryptosystems for keeping information secure in the future.

References

[1] Menon, Vijay. *Deterministic Primality Testing - Understanding The AKS Algorithm.* (2013): arXiv. Web. 1 Sept. 2016.

[2] Klima, Richard E., and Neil P. Sigmon. "Chapter 8. RSA Ciphers." *Cryptology: Classical and Modern with Maplets.* Boca Raton, FL: CRC, 2012. 275-327.

[3] Trappe, Wade, and Lawrence C. Washington. *Introduction to Cryptography With Coding Theory*. 2nd ed. New Jersey: Pearson Education, 2006. Print.

[4] Singh, Simon. *The Alternative History of Public-Key Cryptography.* The Alternative History of Public-Key Cryptography. Cryptome, 6 Oct. 1999. Web. 22 Aug. 2016.

[5] Diffie, Whitefield, and Martin E. Hellman. *New Directions in Cryptology.* Economic and Political Weekly 6.38 (1971): n. pag. Ee.standford.edu. Standford University. Web. 22 Aug. 2016.

[6] Lawson, Thomas. *Odd Orders In Shor's Factoring Algorithm.* Quantum Information Processing 14.3 (2015): 831. Advanced Placement Source. Web. 1 Sept. 2016.

[7] Swami, Balram. *Dual Modulus RSA Based on Jordan-totient Function.* Procedia Technology 24(2016):1581. Web.

[8] The Printable Characters of ASCII and the Corresponding Decimal Codes. 2016. Communications Museum of Macao. Museu Das Comunicaceos. Web. 12 Sept. 2016.

[9] *The Prime Pages*. The University of Tennessee at Martin, n.d. Web. 19 Sept. 2016.

[10] Burton, David M. *Elementary Number Theory*. 5th ed. New York: McGraw-Hill, 2002. Print.

[11] Vani, K. H., and A. G. Aruna. *An Analysis of Quantum Factoring Using Shors Algorithm.* Journal of Network Security Computer Networks 2.1 (2016): n. pag. Journal of Network Security Computer Networks. MAT Journals, 2016. Web. 4 Oct. 2016.

[12] Toffalori, Carlo. *Testing Primality Deterministically.* International Journal Of Quantum Information 3.1 (2005): 269-273. Academic Search Complete. Web. 23 Sept. 2016.

[13] Pommersheim, James E., Tim K. Marks, and Erica Flapan. *Number Theory: A Lively Introduction with Proofs, Applications, and Stories*. Hoboken, NJ: Wiley, 2010. Print.

[14] VALIRON, BENOT, et al. *Programming The Quantum Future. (Cover Story).* Communications Of The ACM 58.8 (2015): 52. Advanced Placement Source. Web. 17 Oct. 2016.

[15] Smolin, John A., Graeme Smith, and Alexander Vargo. *Oversimplifying Quantum Factoring.* Nature 499.7457 (2013): 163. Advanced Placement Source. Web. 17 Oct. 2016.

[16] *The Science of Encryption: Prime Numbers and Mod N Arithmetic* (n.d.): n. pag. Math.berkeley.edu. University of Califonia, Berkeley. Web. 17 Oct. 2016.

[17] Gunsekara, De Vas, Jayathilake, and Perera. *Survey On Prime Numbers.* Reasearch Gate. Department of Mathematics, Faculty of Science, University of Peradeniya, 06 Nov. 2015. Web. 19 Sept. 2016.