Number Theory in RSA Cryptography

Michael Cai, Benjamin Jasper Kra-Caskey, Eric Shao

December 2023

1 Introduction

RSA encryption is universal in modern cryptography. Widely used for data encryption of e-mail and other digital transactions over the Internet, RSA encryption helps keep personal information such as credit card information and messages safe. RSA was developed by three mathematicians, Rivest, Shamir, and Adleman, who used abstract ideas in number theory and modular arithmetic to develop a seemingly simple way of encrypting messages. By doing so, they revolutionized asymmetrical cryptography.

The goal of this paper is to explain the inner workings of RSA including its positive and potential negatives to undergraduate mathematics students proficient in elementary number theory. First, we will delve into how RSA works by introducing modular arithmetic and algorithm complexity, which will help to understand RSA. Then, we will break down how RSA works into four sections: key generation, key distribution, encryption, and decryption. This will encompass the inner workings of RSA. From here, the effectiveness of RSA encryption will be developed through the arguments of asymmetrical encryption and big O notation. By looking at the efficiency of the algorithms used in RSA, we discover that RSA is quite safe against potential classical algorithms. However, quantum algorithms such as Shor's algorithm have the ability to break RSA encryption and transform the world of cryptography. We hope to also explore this method which can crack RSA.

2 Problem Statement

At the heart of RSA encryption is the problem of prime factorization; while large primes are computationally easy to generate and multiply, it is computationally onerous to reverse this process and factor similarly large numbers. This asymmetry is what ensures the security of RSA, making it a vital tool for confidential data transmission. The project explores this aspect by examining key generation and its implications for the encryption and decryption processes. Furthermore, this study will address the impact of quantum computing developments on RSA encryption. The advent of quantum algorithms, such

as Shor's Algorithm, introduces a potential risk to the RSA algorithm, as these could solve the problem of prime factorization on applicable time scales. This evolving landscape necessitates a deep understanding of RSA's mathematical structure and its potential vulnerabilities.

Through this exploration, the project highlights the enduring relevance of RSA encryption in cryptography, defining its role in digital information and the need for advancements in cryptographic methods to stay ahead of computational developments.

3 Objectives

1. Modular Arithmetic and Big O Notation

   Provide an understanding of modular arithmetic, including concepts such as congruence, Euler's Totient Theorem, and modular inverses.

   Explain how big O notation works and how it can be used to analyze algorithms.

2. Comprehension of RSA Algorithm

   Demonstrate clear conceptual understanding of RSA algorithm and describe how it operates.

3. Evaluate RSA Encryption's Security

   Assess the security and effectiveness of RSA encryption, particularly the complexity of factoring large numbers and the efficiency of the asymmetric encryption method.

4. Analyze the Impact of Quantum Computing on RSA

   Describe how Shor's algorithm works and the possible impacts it has on the current world of cryptography.

4 Mathematical Background

4.1 Number Theory

4.1.1 Modular Arithmetic

We define the equivalence relation as follows. In $\mathbb{Z}/n\mathbb{Z}$, we say $a, b \in \mathbb{Z}$ are equivalent, denoted as $a \equiv b \pmod{n}$, iff $n \mid a - b$, or in other words if there exists $k \in \mathbb{Z}$ such that $a - b = nk$.

Addition, multiplication, and exponentiation are defined as they are for the integers.

4.1.2 Euler's Totient Theorem

Define $\varphi(n)$ to be the number of positive integers $k \in \mathbb{Z}^+$, where $k \leq n$ such that $\gcd(k, n) = 1$. Euler's Totient Theorem states that for $a, n \in \mathbb{Z}, \gcd(a, n) = 1$,

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

Proof Consider the set of residues modulo n that are coprime to n, $\{x_1, x_2, \cdots, x_{\varphi(n)}\}$. We claim that the set $S = \{x_1, x_2, \cdots, x_{\varphi(n)}\} = \{ax_1, ax_2, \cdots, ax_{\varphi(n)}\}$, where two elements are equal if they are in the same equivalence class. This will be proven by showing that for all $x \in S$, we have $ax \in S$. Additionally, we will show that for all $x, y \in S, ax \equiv ay \iff x \equiv y \pmod{n}$.

To prove that $x \in S \implies ax \in S$, note that for $x \in S$, we have $\gcd(x, n) = 1$. Then, $\gcd(ax, n) = \gcd(a, n) = 1$, so we must have $\gcd(ax, n) = 1 \implies ax \in S$.

Now consider $x, y \in S, ax \equiv ay \pmod{n} \implies a(x - y) \equiv 0 \pmod{n} \implies n \mid a(x - y)$. However, $\gcd(a, n) = 1$, so $n \mid a(x - y) \implies n \mid x - y \implies x - y \equiv 0 \pmod{n} \implies x \equiv y \pmod{n}$.

Then, the function $f : S \to S$, where $f(x) \equiv ax \pmod{n}$ is shown to be both well-defined and injective. Then, since the cardinality of the codomain equals that of the domain, and f is injective, f must also be bijective. Since f is bijective, $f(S) = S$.

Define $g : \mathcal{P}(n) \to \mathbb{Z}/n\mathbb{Z}$ be defined as $g(A) = \prod_{a \in A} a$ for $A \in \mathcal{P}(n)$. We have $g(f(S)) = a^{\varphi(n)} g(S)$ directly. From $f(S) \equiv S$, we also have $g(f(S)) \equiv g(S)$, so we have $a^{\varphi(n)} g(S) \equiv g(S) \pmod{n}$, so either $a^{\varphi(n)} \equiv 1 \pmod{n}$ or $g(S) \equiv 0 \pmod{n}$. However, all elements in S are coprime to n, so $g(S)$ is coprime to n, so $g(S) \not\equiv 0 \pmod{n}$, so we must have $a^{\varphi(n)} \equiv 1 \pmod{n}$, as desired.

Consequence One consequence of Euler's Totient Theorem is that for $\gcd(a, n) = 1$, when we work with the expression $a^d \pmod{n}$, we can add and subtract any integer multiple of $\varphi(n)$ to the exponent, and the expression remains identical. Then, we can consider equivalence classes when varying only the exponent. Our equivalence classes (not necessarily all distinct) are $a^0, a^1, a^2, \cdots, a^{\varphi(n)-1} \pmod{n}$. We have $a^d \pmod{n} \equiv a^{d+k\varphi(n)}$. Then, we have $x \equiv y \pmod{\varphi(n)} \implies a^x \equiv a^y \pmod{n}$. We can work with the exponent.

Example $5^{100} \pmod{14}$ can be evaluating the fact that since $\gcd(5, 14) = 1$, we have $5^{\varphi(14)} \equiv 5^6 \equiv 1 \pmod{14}$. Then, we have $5^{100} \equiv 5^{6 \cdot 16 + 4} \pmod{14} \equiv (5^6)^{16} \cdot 5^4 \equiv 5^4 \pmod{14} \equiv 9 \pmod{14}$.

Example As another example, if we wish to find $7^{7^7} \pmod{10}$, we note first that $\varphi(10) = 4$. Then, we can add or subtract any multiple of 4 from the exponent. Then, we can replace the exponent, 7^7, with any value congruent

to it modulo 4. Then, we evaluate the lowest residue. We have $7^7 \pmod 4 \equiv (-1)^7 \pmod 4 \equiv -1 \pmod 4 \equiv 3 \pmod 4$. Then, the problem reduces to $7^3 \pmod{10} \equiv 3$.

4.1.3 Multiplicative Inverse

We define the multiplicative inverse (if it exists) of a modulo n to be the value a^{-1} such that $a \cdot a^{-1} \equiv 1 \pmod n$. Note that the multiplicative inverse exists iff $\gcd(a, n) = 1$. Then, we can use Euler's Totient Theorem to describe $a^{-1} \equiv a^{\varphi(n)-1} \pmod n$.

Example Suppose we wish to calculate $7^{-1} \pmod{15}$. We have $\varphi(15) = 8$, so we have $7^{8-1} \equiv 7^7 \equiv 13 \equiv 7^{-1} \pmod{15}$. Indeed, $7 \cdot 13 \equiv 91 \equiv 1 \pmod{15}$.

Using Euler's Totient Theorem here can be computationally slow, since computing $\varphi(n)$ takes many operations. Instead, we use the Euclidean Algorithm, which is a way to both calculate the greatest common divisor of large numbers quickly and also calculate the multiplicative inverse quickly.

4.1.4 Euclidean Algorithm

Suppose we wish to evaluate for $a, b \in \mathbb{Z}^+$, the value $\gcd(a, b)$. We define the sequence of remainders as such. We have $r_0 = a, r_1 = b$, and $r_{i+1} = r_{i-1} - q_i r_i$, where $0 \leq r_{i+1} < r_i$. Note that q is the quotient and r is the remainder. The process boils down to iteratively dividing the remainders and recording the new quotients and remainders. When we have $r_{k+1} = 0$, then the greatest common divisor is the remainder r_k.

We prove now that the Euclidean Algorithm indeed outputs the greatest common divisor. First, note that if for any integers $m, n \in \mathbb{Z}^+$, if we have $m \mid n$, then we must have $m \leq n$. The proof that r_k is the greatest common divisor will be done by first showing that $\gcd(a, b) \mid r_k$, and then showing that $r_k \mid \gcd(a, b)$, so the two must be equal.

Let g be the greatest common divisor of a and b. We can then write $a = gx$ and $b = gy$. Without loss of generality let $a > b$. Then, $a - b = g(x - y)$. More generally, for r_i and r_{i+1}, we have $r_{i+1} = r_{i-1} - q_i r_i$. If g is the greatest common divisor of r_{i-1} and r_i, g must still divide r_{i+1} after the subtraction. By repeatedly applying this argument, we know that for the last non-zero remainder r_k, we have $g \mid r_k$, so $g \leq r_k$.

We now show that $r_k \mid a$ and $r_k \mid b$. Note that since $r_{k+1} = 0$, we have $r_{k+1} = 0 = r_{k-1} - q_k r_k \implies q_k r_k = r_{k-1}$, so r_k divides r_{k-1}. Consider the next equation $r_{k-2} = q_{k-1} r_{k-1} + r_k$. Since r_k divides both terms on the right hand side, it must also divide the left hand side, so $r_k \mid r_{k-2}$. This argument can be repeated, resulting in $r_k \mid r_0 = a$ and $r_k \mid r_1 = b$, so r_k is a divisor of both a and b, so it must be a divisor of $\gcd(a, b) = g$, so we have $r_k \mid g \implies r_k \leq g$,

since both r_k, g are non-zero. Then, since we have $g \leq r_k$ and $r_k \leq g$, we must have $r_k = g$.

Example Suppose we wish to find $\gcd(210, 45)$. We write the equations of quotients and remainders. We have $r_0 = 210$ and $r_1 = 45$. Now, we have $r_2 = r_0 - q_1 r_1$. In order to ensure that $0 \leq r_2 < r_1$, we have $q_1 = 4$, so we have $r_2 = 210 - 4 \cdot 45 = 30$. Then, we have $r_3 = r_1 - q_2 r_2$. To have $0 \leq r_3 < r_2$, we have $q_2 = 1$. Then, we have $r_3 = 45 - 30 = 15$. Then, we have $r_4 = r_2 - q_3 r_3$. To have $0 \leq r_4 < r_3$, we must have $q_3 = 2$. Then, we have $r_4 = 30 - 2 \cdot 15 = 0$. Now, we stop, and the greatest common divisor is $r_3 = 15$.

Finding the Multiplicative Inverse Quickly The Euclidean Algorithm can also be used to calculate the multiplicative inverse quickly. Suppose for $\gcd(a, n) = 1$, we wish to find $a^{-1} \pmod{n}$ satisfying $aa^{-1} \equiv 1 \pmod{n}$. We have the system of equations, with $n = r_0, a = r_1$,

$$\begin{aligned} r_2 &= r_0 - q_1 r_1 \\ r_3 &= r_1 - q_2 r_2 \\ r_4 &= r_2 - q_3 r_3 \\ &\vdots \\ r_{k-1} &= r_{k-3} - q_{k-2} r_{k-2} \\ r_k &= r_{k-2} - q_{k-1} r_{k-1} = \gcd(a, b) = 1 \end{aligned}$$

We take the last equation $r_{k-2} - q_{k-1} r_{k-1} = 1$ and substitute in our equation for r_{k-1} in terms of r_{k-2} and r_{k-3}. Then, we substitute r_{k-2} in terms of r_{k-3} and r_{k-3} and r_{k-4}. These substitutions can be repeated until our the right hand side of our initial equation, 1, is expressed entirely in terms of $r_0 = n$ and $r_1 = a$. Then, our equation becomes $r_0 x + r_1 y = nx + ay = 1$ for $x, y \in \mathbb{Z}$. Taking the corresponding equation modulo n yields $ay \equiv 1 \pmod{n}$. Then, the multiplicative inverse is the computed coefficient y, found by substituting the equations from the Euclidean Algorithm.

Example Suppose we wish to find $x, y \in \mathbb{Z}$ satisfying $14x + 3y = 1$. Our system of equations is $r_0 = 14, r_1 = 3$,

$$\begin{aligned} r_2 &= r_0 - q_1 r_1 = 14 - 4 \cdot 3 = 2 \\ r_3 &= r_1 - q_2 r_2 = 3 - 1 \cdot 2 = 1. \end{aligned}$$

Then, we have $q_1 = 4, q_2 = 1$. We take the final equation, $r_1 - q_2 r_2 = 1$ and substitute in $r_2 = r_0 - q_1 r_1 = r_0 - 4r_1$. Then, we have $1 = r_1 - q_2 r_2 = r_1 - q_2(r_0 - q_1 r_1) = (q_1 q_2 + 1) r_1 - q_2 r_0 = 5r_1 - r_0 = 5 \cdot 3 - 14$. Then, we have $5 \cdot 3 \equiv 1 \pmod{14}$, so we have $3^{-1} \equiv 1 \pmod{14}$.

4.2 Time Complexity

Big O notation is a way to describe how a function $f(x)$ grows as $x \to \infty$, and is commonly used to describe the time complexity of an algorithm. We denote $f(x) = O(g(x))$ if there exists $M \in \mathbb{R}^+$ such that $f(x) \leq Mg(x), \forall x \geq x_0$ for some $x_0 \in \mathbb{R}^+$. For instance, for $f(x) = 2x^2 + 3x + 2$, we can choose $g(x) = x^2, M = 3, x_0 = 4$, and we have $Mg(x) \geq f(x), \forall x \geq x_0 = 4$, so we can write $f(x) = O(x^2)$. In general, a polynomial of degree n is $O(x^n)$. Certain sorting algorithms such as merge sort and quick sort take $O(n \log n)$ time. In general, big O notation describes the fastest growing term of a function, or bounds the function by a fastest growing term.

Big O notation effectively describes how fast a function grows in the worst case scenario. The time taken to execute an algorithm can be estimated by plugging in the input into the function in Big O notation. While this may be off by a constant factor, this constant factor is typically negligible for large inputs. Typically, a single 1 GHz computer can perform 10^8 operations per second.

To describe the time complexity of an algorithm, we calculate, based on the input, the number of total operations needed to complete the algorithm in terms of the input parameter sizes. For a single-input function $f(x)$ denoting the number of needed operations, we find $g(x)$ where $f(x) = O(g(x))$, and for the "smallest" $g(x)$, we write that as the time complexity.

5 Main Results: How RSA Works

RSA consists of four major steps: key generation, key distribution, encryption, and decryption. The goal of RSA is to encrypt messages in a way such that even if they are intercepted, the message can not be easily decrypted. For secure communication, RSA uses a public key and a private key, in which the public key is known to everyone. The public key consists of the two integers, the modulus n and the public exponent e. The private key $d \in \mathbb{Z}$ is kept a secret, and is found ahead of time by solving the following:

$$(m^e)^d \equiv m \pmod{n}$$

where m here is the message being sent and $n = pq$ for two large prime numbers p, q. Note that to find d such that $(m^e)^d = m^{ed} \equiv m \pmod{n}$, it suffices to have $ed \equiv 1 \pmod{\varphi(n)}$. For $n = pq$, the expression $\varphi(n) = \varphi(pq)$ evaluates to $(p-1)(q-1)$. Then, to find a suitable decryption key d, we find d such that $ed \equiv 1 \pmod{(p-1)(q-1)}$.

5.1 Key Generation

RSA relies on the fact that it is computationally easy to find large prime numbers, but computationally expensive to find all the prime factors of a similarly large number. The public key for RSA consists of two integers, n and e, while

the private key consists of d, as well as everything that could be used to calculate d, namely p, q, and $\lambda(n)$. The Carmichael function $\lambda(n)$ is used as an extension of Euler's totient function $\varphi(n)$. For simplicity, this paper will continue to use $\varphi(n)$, as RSA still functions when only using φ.

To generate the public key integer modulus n, the encrypter generates two very large prime numbers, p and q, generally through randomly generating large integers until both are prime. The encrypter then multiplies them together to get an even larger number n. p and q are kept private, and because it is computationally difficult to decompose large numbers to their prime factors, n can be made public without p and q becoming easy to discern. Generally, p and q are chosen to be roughly half the size of n; that is, if n is 1024 bits, p and q are each around 512 bits. The encrypter then applies evaluates $\varphi(n) = \varphi(pq) = (p-1)(q-1)$, which is kept private.

Next, the encrypter generates e, the last part of the public key. To do this, they choose an integer $2 < e < \varphi(n)$ such that e and $\varphi(n)$ are coprime to each other. Finally, as explained previously, the encrypter generates d, the last part of the private key, as the modular multiplicative inverse of e modulo $\varphi(n)$, satisfying $ed \pmod{\varphi(n)} \equiv 1$. This generation of d is generally done using the extended Euclidean algorithm. Although it can also be done with modular exponentiation, that would be much slower.

5.2 Key Distribution

For the purpose of distributing keys, let us have two people that want to communicate with each other and let us name them Alice and Bob. Suppose that Bob wants to send a message to Alice. Alice will create a key-pair, (n, e), which is the public key that Alice will upload to a server for others to view. Alice will also generate the private key (d), which is kept private.

5.3 Encryption

Now suppose that Bob has attained Alice's public key and wants to send a message M to Alice. Firstly, the message is converted into plain text represented as integer m. Since Bob knows the public key (n, e), Bob evaluates and sends back to Alice the encoded message

$$c \equiv m^e \pmod{n},$$

which is evaluated easily using modular exponentiation. In order for the message to be unambiguous, m must be less than n, which puts a constraint on the message length using this method.

5.4 Decryption

After Alice receives the value c from Bob, Alice can recover the message m from c by computing the following:

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

Where Alice knows d by solving

$$de \equiv 1 \pmod{(p-1)(q-1)}$$

by using the Euclidean Algorithm.

5.5 Effectiveness of RSA Encryption

The crux of RSA's effectiveness relies on the difficulty of computing a decryption key d from the public key n. $\varphi(n)$ must first be evaluated, which can be done naively by either iterating through the integers from 1 to n, or by trying to factor $n = pq$ into its two prime factors. Factoring n by iterating through integers from 2 to $\sqrt{n}$ takes $O(\sqrt{n})$ time, which is referred to as exponential time (an exponential function of $\log n$, the number of digits of n). In contrast, finding two large primes p, q takes $O(\log n)$ time, and the decryption key can be computed by Alice in $O(\log n)$ time. Then, the entire message encryption and decryption process takes $O(\log n)$ time, which is polynomial time (polynomial of $\log n$, the number of digits), whereas intercepting the message takes $O(\sqrt{n})$ time, which is exponential time. For this reason, extremely large values of n (up to 2^{2048}) are used, so the messaging process takes roughly 2000 operations, while interception with naive factoring takes roughly 2^{1024} operations, which is infeasible.

Because of this major computational constraint, classical computing algorithms do not provide viable options to intercept messages encrypted by RSA. In spite of this, mathematical research has provided a feasible approach for factoring large primes through study of quantum algorithms which can massively reduce this computational burden, endangering the security of RSA. This quantum approach is Shor's algorithm.

6 Breaking RSA Encryption: Shor's Algorithm

The bottleneck slowing down the interception of RSA messages is the $O(\sqrt{n})$ time taken to factor n to find the decryption key. Hypothetical algorithms designed to intercept these messages revolve around finding more efficient ways to factor large $n = pq$. Shor's algorithm is able to decompose any number into its prime factors in a reasonable time scale.

6.1 Classical Reduction

In classical reduction, our goal is to develop a method to factor any arbitrary number N into two numbers, p and q, which are both greater than one. If we

can find a method to do this, we can recursively apply this method until we have all the prime factors of N.

First, we have methods to trivially reduce N if it is even, as 2 would be a prime power and we can apply Classical reduction to $N/2$, as well as if it is a prime power, for which we have other methods to classical reduce N. Because of this, we can assume that N is an odd number which isn't a prime power. Next, we want to randomly pick a number a such that $2 \leq a < N$, and find the greatest common divisor between N and a using Euclid's algorithm. If the greatest common divisor $\neq 1$, we can stop, as we have found a factor for N, since $\gcd(N, a)$ and $\frac{N}{\gcd(N,a)}$ are factors for N. If $\gcd(N, a) \neq 1$, then we apply the quantum subroutine, an algorithm which, given that a and N are coprime, returns r such that $a^r \equiv 1 \pmod{N}$. Since $a^r \equiv 1 \pmod{N}$, we can see that $a^r - 1 \pmod{N} \equiv 0$; that is, $a^r - 1$ divides by N. By difference of squares, we can see that this means that $(a^{r/2} - 1)(a^{r/2} + 1)$ divides by N. If r is odd, then $a^{r/2}$ is not an integer, so $(a^{r/2} - 1)$ and $(a^{r/2} + 1)$ aren't integers and thus don't divide by N, so we have to restart, picking a new value for a.

Otherwise, we can assume that r is even and thus calculate $g \equiv \gcd(N, a^{r/2}+1)$. If $g = 1$, then we can't find factors for N and have to pick a new value for a. Otherwise, g and $\frac{N}{g}$ are two factors of N, satisfying our goal, to find two integers p and q such that $pq = N$. Thus, through recursively repeating this method we can find all the prime factors of N.

6.2 Quantum order-finding subroutine

Through application of Fourier and Hadamard transformations to put K qubits into an equal superposition over all Q basis states, where $Q = 2^K$, it is possible to find the lowest integer r such that $a^r \equiv 1 \pmod{N}$ for any integer N where $N^2 \leq Q \leq 2N^2$. Unfortunately, to limit the length of this paper, we won't go into depth on the math behind this, as it requires a deep technical understanding of both Fourier/Hadamard Transformations as well as quantum computing. And since our current freshman undergraduate course to whom we are writing this to will not have sufficient knowledge, we will continue on without detailing this. What's important about the quantum order-finding subroutine is that it allows us to find the lowest r such that $a^r \equiv 1 \pmod{N}$.

6.3 Effectiveness of Shor's Algorithm

Shor's algorithm for factoring $n = pq$ has a time complexity of

$$O((\log n)^3(\log\log n)(\log\log\log n))$$

(Li et al.), which is considered polynomial time (a polynomial of $\log n$, the number of digits).

For $n \approx 2^{2048}$, there are roughly $3 \cdot 10^{11}$ operations performed to compute a decryption key, which is likely feasible for a quantum computer and thus

threatens the security of RSA encryption.

7 Discussion

7.1 Mathematical Results

In the context of the goal of this paper, the use of an asymmetrical encryption method (RSA) provides a robust method of sending and receiving information. By computing two positive integers p and q that are said to be relatively prime, we can generate the private key. We then generate the public key by computing $n = pq$ and integer e such that $2 < e < \varphi(n)$ such that e and $\varphi(n)$ are co-prime.

Example For simplicity sake, choose relatively small prime integers $p = 3$ and $q = 11$. From here we can compute $n = pq = 3 \cdot 11 = 33$. Now compute $\varphi(n) = (p-1)(q-1) = 2 \cdot 10 = 20$. From here, we can choose e such that $2 < e < \varphi(n)$ where e and $\varphi(n)$ are coprime. Therefore, let $e = 7$. Now we can compute the value for d such that $de \pmod{\varphi(n)} \equiv 1$. One solution would be $d = 3$, since $3 \cdot 7 \pmod{20} \equiv 1$. Thus, the public key is $(e, n) = (7, 33)$ and the private key is $d = 3$. The encryption of $m = 2$ is $c \equiv 2^7 \pmod{33} \equiv 29$ and the decryption of $c = 29$ is $m \equiv 29^3 \pmod{33} \equiv 2$.

It is obvious that with small integers p and n that it is possible to break RSA encryption with brute force attacking methods such as simply factorizing n by testing all digits $1 < p < \sqrt{n}$. Thus, p and q are chosen such that n is at least 1024 bits. Therefore, RSA algorithms operate with large algorithms that involves computationally expensive operations when compared to operations involved in other single-key systems such as the Data Encryption Standard.

The quantum subroutine which Shor's Algorithm relies on is impossible for classical computing yet doable for quantum computers. This makes quantum computers a possible threat to RSA, as with quantum computers completing this quantum order-finding subroutine, Shor's Algorithm could be a viable method used to crack RSA encryption.

Despite this possibility, there are some constraints on the application of the Quantum order-finding subroutine, based on the amount of qubits necessary to factor a large number. For any modulus n that the encrypter chooses, K qubits are required such that $2^K \geq n^2$. Although this would not cause a problem for classical computers, the largest quantum computer contains around 1125 qubits, which limits the maxiumum decryptable value for n at $\sqrt{2^{1125}}$, or 2^{562}. While impressive, this is nowhere near able to crack the n values currently used, which range from 2^{2048} all the way to 2^{4096}, and make RSA secure for the near future until 8192-qubit quantum computers are developed.

IBM, an industry leader who broke the 1000 qubit barrier in 2023, currently has plans to complete the largest quantum computer ever made in 2025, which would contain over 1300 qubits. Assuming this rate of growth is exponential,

it will take around 15 years for quantum computers to be able to crack RSA encryption currently used in industry, which gives industries which use RSA a buffer to shift away from RSA to more quantum-safe encryption.

7.2 Applications and Implications

RSA cryptography has huge applications in our modern world, as RSA is used for secure communications, digital signatures, email encryption, file encryption, secure code signing, secure authentication, secure remote access, secure transactions, and much much more.

With digital signatures, RSA provides a way to verify the authenticity of digital documents by providing a private key to encrypt the hash (a signature that identifies some amount of data, usually a file or message) of a document, resulting in the validity of opening the public key.

Email and file encryption also use RSA for encrypting the messages and attachments such as images, videos, and more. This ensures that only the intended recipient of the file or email can read the document and protects them from unauthorized use.

Needless to say, RSA encryption is a major player in the world of asymmetric encryption schemes and provides valuable and wide variation in the fields of information security, protecting sensitive information such as SSN, bank account info, confidential buisness files, credit card details, and more. Because of this reliance on RSA, breaking RSA would completely transform the field of cryptography, and there is a race for methods to break RSA encryption. One possible method for this is quantum computing. Currently, the National Science Foundation invests 38 million dollars per year in quantum research. This agency has awarded 22 separate grants as part of the its investment into physics, computer sciences, materials research, engineering and chemistry, all to develop quantum computers. Additionally, large corporations such as IBM, Microsoft, and Nvidia have launched initiatives to develop the first viable quantum computer (Castelvecchi).

When quantum computers are developed which can decrypt RSA, it will mark the end of RSA encryption as a method for secure communication. This understanding of the weakness of RSA has intensified the search for alternative encryption methods which are quantum-safe, a field which will only continue to grow as better quantum computers are developed.

8 Conclusion

In this paper, the methodology behind RSA encryption is explained, and modular arithmetic is used to explain its validity. Because of the time complexity of decryption compared to message interception, RSA is a relatively simple and very effective message sending process. Additionally, because of the bottleneck

of factoring the decryption key, with sufficiently large n RSA is essentially unbreakable by conventional computing.

This paper also discusses the methodology behind Shor's algorithm, a quantum algorithm for prime factorization which is made more and more feasible by the advent of quantum computers, and how it threatens the security of RSA encryption through its ability to decrypt encrypted messages in polynomial time.

RSA Encryption is a pillar of modern cryptography. Relying on the fact that it is very easy to find large primes but extremely hard to decompose large numbers to their prime factors, RSA allows us to encrypt a message that no one besides the target of the message, including the encrypter themselves, can decrypt. This type of asymmetric encryption is the basis for security on a public server such as the internet, and enables much of our modern world, including much of the banking industry, secure browsing such as HTTPS, and the privacy of emails.

As a result of the rise in quantum computing and its threat to the safety of RSA encryption, alternative encryption methods are being sought that are safe from quantum development. One such example includes cryptography based on the lattices formed by different basis vectors. Regardless, RSA's effect on modern cryptography is monumental.

9 References

"Art of Problem Solving." Artofproblemsolving.com, 2018, https://artofproblemsolving.com/wiki/index.php/Euler%27s_Totient_Theorem.

"Big O Notation." Wikipedia, Wikimedia Foundation, 7 Dec. 2023, https://en.wikipedia.org/wiki/Big_O_notation.

Castelvecchi, Davide. "IBM Releases First-Ever 1,000-Qubit Quantum Chip." Nature, 4 Dec. 2023, www.nature.com/articles/d41586-023-03854-1, https://doi.org/10.1038/d41586-023-03854-1.

"Euclidean Algorithm." Wikipedia, Wikimedia Foundation, 1 Dec. 2023, https://en.wikipedia.org/wiki/Euclidean_algorithm.

"Extended Euclidean Algorithm." Wikipedia, Wikimedia Foundation, 24 Nov. 2023, https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm.

Jonsson, Jakob, and Burt Kaliski. "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1." IETF, 1 Feb. 2003, https://datatracker.ietf.org/doc/rfc3447/.

Landquist, Eric. The Quadratic Sieve Factoring Algorithm. 2001. https://www.cs.virginia.edu/crab/QFS_Simple.pdf

Li, Jun, et al. "An Efficient Exact Quantum Algorithm for the Integer Square-Free Decomposition Problem." Scientific Reports, vol. 2, no. 1, 10 Feb. 2012, p. 260, www.nature.com/articles/srep00260, https://doi.org/10.1038/srep00260.

Rivest, R. L., et al. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." Communications of the ACM, vol. 21, no. 2, 1978, pp. 120–126, https://people.csail.mit.edu/rivest/Rsapaper.pdf, https://doi.org/10.1145/359340.359342.

"RSA (Cryptosystem)." Wikipedia, Wikimedia Foundation, 10 Dec. 2023, https://en.wikipedia.org/wiki/RSA_(cryptosystem).

Shor's Order (Period) Finding Algorithm and Factoring. 2005, https://inst.eecs.berkeley.edu/~cs191/fa05/lectures/lecture19_fa05.pdf

Shor, Peter W. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer." SIAM Review, vol. 41, no. 2, Jan. 1999, pp. 303–332, https://arxiv.org/pdf/quant-ph/9508027, https://doi.org/10.1137/s0036144598347011.