THE RSA CRYPTOSYSTEM

JOSHUA PITÉ AND YIYING ZHONG
MENTOR: HONGLIN ZHU

Abstract. In the field of cryptography, which aims to ensure secure message communication, the RSA public key cryptosystem is the oldest widely applied secure data transmission method. In this expository paper, we provide a historical and technical overview of the RSA cryptosystem. We introduce the mathematical methods used in RSA, present the steps of the algorithm, discuss complexity results relating to the security of RSA, and implement a Python version of RSA.

1. Introduction

The very earliest history of pre-computer cryptography dates back to the time humans began using written communication. Before computers were invented, people tended to choose ciphers to encrypt and decrypt the messages. One famous example of such communication is the Caesar cipher, used by Julius Caesar around 58 B.C. [6]. The Caesar cipher, also known as the shift cipher, was a substitution method to shift letters a fixed number of positions down the alphabet[1], which could make the message unreadable without decryption. However, the Caesar cipher was not a secure way to encrypt the messages. In our daily communication, certain letters will be used more frequently than others. Comparing the average frequency of each letters in daily communication with the frequencies in the encrypted messages being sent makes it easy to determine the correlations between plain letters and cipher letters.

Later in the Middle Ages, with the invention of cryptanalysis, the simple substitution cipher was no longer secure, prompting further development in both cryptography and cryptanalysis. From homomorphic ciphers to polyalphabetic ciphers, humans started to enhance the security level by using multiple substitutes for each letter. Due to their ability to keep information relatively safe from outsider's interpretation, those ciphers and codes have been commonly used in the military and for political affairs since the 18th century. The Second Industrial Revolution advanced cryptography and cryptanalysis to an even higher level. While the military could communicate more efficiently using radios and telegrams, the messages, however, were at higher risk of being interfered or decrypted by the enemy. In order to address the issues that arose with radio communication, countries invented different encryption machines to create incredible complex polyalphabetic ciphers, for example, the Enigma machine with multiple rotors and the Purple machine using switches.

Then, with the development of computer cryptography, mathematicians and computer scientists invented two kinds of cryptography: private key cryptography and public key cryptography [4]. In private key cryptography, the private key is shared between the sender and the receiver, and is used in both encryption and decryption. Public key cryptography requires one public key, which is published for encryption, and one private key, which is kept

[1]The shift is the same fixed number for every letter.

secret for decryption, which can better enhance the security level of the communication. The greatest difference between private and public cryptosystems is the foundational method of keeping the communication secure. The security of a private cryptosystem relies on the secret private key. With the pre-exchanged secret key also known to the two communicating parties, their following messages can be considered secure to a certain degree. The security of a public cryptosystem depends on the strength of the algorithm and the difficulty of obtaining the private key used for decryption.

In 1976, Whitfiled Deffie and Martin Hellman published a paper, [2], about their ground-breaking key exchange method, which promoted the invention of other public key cryptosystem methods. Deffie-Hellman key exchange algorithm helps the two communicating parties to exchange their shared keys in an insecure setting. Though the fascinating Diffie-Hellman key exchange algorithm helps exchange the private keys safely, it cannot be used to send messages for communication. RSA public key cryptosystem actually achieved that goal. Named after its three public inventors: Ron Rivest, Adi Shamir and Leonard Adleman[2], RSA public key cryptosystem is the first invented public key cryptosystem and one of the most commonly one in the world [5].

RSA cryptosystem has two public keys published for encryption, and one private key only known to the receiver for decrypt the message. Also, the RSA cryptosystem shares the similar feature of public key cryptosystems, where its security depends on the algorithm problem, instead of the secret pre-exchanged private keys. Hence, the RSA cryptosystem allows the communicating parties to directly transmit the message safely in an insecure channel without sharing the private keys beforehand. In this paper, we will dive deeper into the mystery of RSA cryptosystem and learn how its algorithm works.

1.1. **Outline.**

We now present an overview for better understanding the structure of our writing. The rest of the paper is outlined as follows. In section §2, we introduce the relevant mathematical background. In section §3, we break down the RSA algorithm used for its cryptosystem, section §4, we talk about multiple reasons why RSA cryptosystem is considered secure, and in last section §5, we present some common implementation methods of the RSA algorithm.

2. Mathematical preliminaries

The mathematical foundation of cryptography involves the field of number theory. In this section, we give an overview of the relevant mathematical background for an analysis of the RSA cryptosystem.

2.1. **Prime numbers.**

A major object of study in number theory is the set of prime numbers. An integer p is a *prime number*, if $p \geq 2$, and the only positive integers dividing p are 1 and p.

While integers are also commonly studied in number theory, we may not always have one integer divides another, which leads to the concept of divisibility. Given two integers a and b, with $b \neq 0$, we say that b *divides* a, or that a *is divisible by* b if there is an integer c such that

$$a = bc.$$

[2]All three where at MIT in 1977.

We write $b \mid a$ to say that b divides a. If b does not divide a, then we denote it as $b \nmid a$.

A *common divisor* of two integers a and b is a positive integer that divides both a and b. A *greatest common divisor* is the greatest positive integer that divides a and b. We denote it as $\gcd(a, b)$. If we have two positive integers a and b, and their greatest common divisor is 1, $\gcd(a, b) = 1$, then we say these two positive integers a and b are *co-prime* or *relatively prime*. As the numbers become larger, it will be harder to find their greatest common divisor by simply listing their divisors separately and finding the largest shared one. One efficient method to find the greatest common divisor is based on a division algorithm.

2.2. **The Euclidean algorithm.**

Lemma 2.1 (Division algorithm)**.** *Given two positive integers a and b, there exist unique integers q and r such that*

$$a = b \cdot q + r \quad \textit{with} \quad 0 \le r < b.$$

Here, q is called the quotient *and r is called the* remainder*.*

We now present the Euclidean algorithm which finds the greatest common divisor of two positive integers.

Algorithm 2.2 (Euclidean algorithm)**.** Let a and b be positive integers. We first apply the division algorithm on a and b to get

$$a = b \cdot q + r \quad \text{with} \quad 0 \le r < b.$$

If d is any common divisor of a and b, then $a = k_1 \cdot d$ and $b = k_2 \cdot d$. Because $r = a - b \cdot q = (k_1 - q \cdot k_2) \cdot d$, d must also divide r. Hence, if e is a common divisor of b and r, then e will be a divisor of a. In other words, the common divisors of a and b are the same as the common divisors of b and r. So

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

Repeating the process, we divide b by r and get another similar equation, which is

$$b = r \cdot q' + r' \quad \text{with} \quad 0 \le r' < b,$$

in which we will also have

$$\gcd(b, r) = \gcd(r, r').$$

We can repeat the process of dividing by the reminder, and the resulting reminders will get smaller until it reaches zero. At that point, with the second to the last reminder s, $\gcd(s, 0) = s = \gcd(a, b)$, and we complete the process to find the greatest common divisor of a and b.

The general method of such way of finding the greatest common divisor is also called *Euclidean Algorithm.*

Theorem 2.3 (Extended Euclidean algorithm)**.** *Let a and b be two positive integers, then there always be integers u and v satisfying this equation*

$$au + bv = \gcd(a, b).$$

Proof. Looking back at the process of finding the greatest common divisor in Algorithm 2.2, we know we can get a series of equations

$$\begin{aligned}
a &= b \cdot q_1 + r_1 && \text{with } 0 \le r_1 < b,\\
b &= r_1 \cdot q_2 + r_2 && \text{with } 0 \le r_2 < r_1,\\
r_1 &= r_2 \cdot q_3 + r_3 && \text{with } 0 \le r_3 < r_2,\\
&\vdots && \vdots\\
r_{n-2} &= r_{n-1} \cdot q_n + r_n && \text{with } 0 \le r_n < r_{n-1},\\
r_{n-1} &= r_n \cdot q_{n+1} + 0.
\end{aligned}$$

Now rewriting r_n in terms of the previous r, we have

$$\begin{aligned}
r_n &= r_{n-2} - r_{n-1} \cdot q_n,\\
r_{n-1} &= r_{n-3} - r_{n-2} \cdot q_{n-1},\\
r_{n-2} &= r_{n-4} - r_{n-3} \cdot q_{n-2},\\
&\vdots\\
r_2 &= b - r_1 \cdot q_2,\\
r_1 &= a - b \cdot q_1.
\end{aligned}$$

Substituting r_{n-1} into r_n and rearranging the terms, we get

$$\begin{aligned}
r_n &= r_{n-2} - (r_{n-3} - r_{n-2} \cdot q_{n-1}) \cdot q_n\\
&= r_{n-2} \cdot (1 + q_{n-1} \cdot q_n) + r_{n-3} \cdot q_n
\end{aligned}$$

Conducting the same process once again with r_{n-2}, we get

$$\begin{aligned}
r_n &= (r_{n-4} - r_{n-3} \cdot q_{n-2}) \cdot (1 + q_{n-1} \cdot q_n) + r_{n-3} \cdot q_n\\
&= r_{n-3} \cdot (q_n - (q_{n-2}) \cdot (1 + q_{n-1} \cdot q_n)) + r_{n-4} \cdot (1 + q_{n-1} \cdot q_n)
\end{aligned}$$

Keep substituting previous r_i values and combining the terms, we can finally achieve this equation $r_n = a \cdot u + b \cdot v$, where u and v are two integers. With $\gcd(a, b) = \gcd(r_n, 0) = r_n$ also known from Algorithm 2.2, we completed the proof for $\gcd(a, b) = au + bv$. ☐

2.3. **Modular arithmetic.**

Another commonly used tool in cryptography is modular arithmetic and its congruence.

Let $m \ge 1$ be an integer. The integers a and b are *congruent modulo* m if their difference $a - b$ is divisible by m. We write it as

$$a \equiv b \pmod{m}.$$

The number m is called the *modulus.*

Corollary 2.4. *Let $m \ge 1$ be an integer, and a is an integer as well, then*

$$a \cdot b \equiv 1 \pmod{m}$$

for some integer b if and only if $\gcd(a, m) = 1$. *If such an integer b exists, then we say b is the* multiplicative inverse *of a modulo m.*

Proof. First suppose $\gcd(a, m) = 1$. Based on Theorem 2.3, there will be two integers u and v satisfying the equation $au + mv = \gcd(a, m) = 1$. It can also be written as $au - 1 = -mv$. Hence, $au \equiv 1 \pmod{m}$, and we can take u as b.

For the other way, suppose that a has an inverse modulo m, so $a \cdot b \equiv 1 \pmod{m}$. This means there is some integer c satisfy $ab - 1 = cm$. Using Theorem 2.3 again, $\gcd(a, m) = ab - cm = 1$, which completes our proof of the corollary: a has an inverse modulo m if and only if $\gcd(a, m) = 1$. □

We now state a fundamental result in elementary number theory, Fermat's little theorem.

Theorem 2.5 (Fermat's little theorem)**.** *Let p be a prime number and a be an integer such that $p \nmid a$. Then*

$$a^{p-1} \equiv 1 \pmod{p}.$$

Proof. First look at a list of numbers

$$a, 2a, 3a, 4a, \ldots, (p-1)a \quad \text{reduced modulo} \quad p. \tag{2.1}$$

We first show that these $p - 1$ numbers are all different modulo p. Suppose, for the sake of contradiction, that we have two numbers from the list ja and ka such that

$$ja \equiv ka \pmod{p}, \quad \text{so that} \quad (j-k)a \equiv 0 \pmod{p}.$$

Since $p \nmid a$, we must have $p \mid (j - k)$, but $-(p - 2) \leq j - k \leq p - 2$, hence $j - k = 0$, and we prove the assumption that these numbers in the list are all different.
Now to prove the theorem, we multiply all the numbers in the list (2.1) together and reduce modulo p, which gives

$$a \cdot 2a \cdot 3a \cdots (p-1)a \equiv 1 \cdot 2 \cdot 3 \cdot (p-1) \pmod{p}.$$

This can also be written as

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

With $p \nmid (p - 1)!$, we can cancel $(p - 1)!$ at both sides, which gives us

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

and we completed the proof of Fermat's little theorem. □

2.4. **Euler's theorem.** We need two more ingredients, Euler's totient function and Euler's theorem, which is a generaliation of Fermat's little theorem.

Definition 2.6. Euler's totient function, $\phi(n)$, counts the positive integers up to a given integer n that are relatively prime to n. That is, $\phi(n)$ is the number of $m \in \mathbb{N}$ such that $1 \leq m < n$ and $\gcd(m, n) = 1$.

Example 2.7. Here are values of $\phi(n)$ for the first few positive integers:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\phi(n)$	1	1	2	2	4	2	6	4	6	4	10	4	12	6	8

Proposition 2.8. *(i) If p is a prime number, then $\phi(p) = p - 1$.*
(ii) If p and q are distinct prime numbers, then $\phi(pq) = (p - 1)(q - 1)$.

Proof. (i) If p is a prime number, then $\gcd(p, k) = 1$ for all integer $1 \leq k < p$, so we have $p - 1$ numbers that are relatively prime to p. Hence, $\phi(p) = p - 1$.

(ii) Since the proper divisors of pq are 1, p, and q, any integer n with $1 \leq n < pq$ must have $\gcd(n, pq)$ being one of these three numbers. Complementary counting gives that
$$\phi(pq) = pq - 1 - (q-1) - (p-1) = (p-1)(q-1).$$
$\square$

We now state Euler's theorem.

Theorem 2.9 (Euler)**.** *If n and a are coprime positive integers, then $a^{\phi(n)}$ is congruent to 1 modulo n, where ϕ denotes Euler's totient function; that is*
$$a^{\phi(n)} \equiv 1 \pmod{n}.$$

Remark 2.10. The proof of Theorem 2.9 can be found in [5]. However, we will only need the statement in the special case when $n = pq$ is a product of two distinct prime numbers. This case follows from Theorem 2.5 and Proposition 2.8. Indeed, we have that
$$a^{\phi(pq)} = a^{(p-1)(q-1)}$$
is congruent to 1 modulo p and modulo q. Hence, $a^{\phi(pq)} - 1$ is a multiple of both p and q, hence a multiple of pq. Therefore,
$$a^{(p-1)(q-1)} \equiv 1 \pmod{pq}.$$

Example 2.11. Euler's theorem may be used to easily reduce large powers modulo n. For example, consider finding the ones place decimal digit of 7^{222}, i.e. $7^{222} \pmod{10}$. The integers 7 and 10 are coprime, and $\phi(10) = 4$. So Euler's theorem yields $7^4 \equiv 1 \pmod{10}$, and we get $7^{222} \equiv 7^{4\times 55+2} \equiv (7^4)^{55} \times 7^2 \equiv 1^{55} \times 7^2 \equiv 49 \equiv 9 \pmod{10}$.

3. RSA public key cryptography

To better illustrate the RSA public key cryptography, we consider a scenario in which Alice wants to send a secret message m to Bob, but Eve can intercept all communications between the two. Therefore, they must engage in some encryption and decryption protocol so that Eve cannot obtain the message. The steps of the RSA algorithm are shown in Table 1.

In order to set up this public key cryptosystem, Bob chooses two large[3] prime numbers, p and q. Using these two numbers, Bob computes his public key pair (N, e), with the *modulus* $N = pq$, and the *encryption exponent* e, which is an integer co-prime to $(p-1)(q-1)$. Then Bob sends this key pair (N, e) to Alice.

When Alice wants to communicate with Bob, she would covert her message into an integer, the *plaintext* m. She then uses the public key pair (N, e) to compute $c \equiv m^e \pmod{N}$. Alice sends this *ciphertext* c to Bob.

It is important to note that any message being sent may have the risk of being intercepted by Eve, so not only the modulus N, encryption exponent e are known, the ciphertext c may be known to Eve as well.

We now show how Bob can recover the message m. In §4, we discuss why it is hard for Eve to obtain m with only N, e, and c.

[3]In practice these numbers are 200 to 600 decimal long.

Alice	**Eve**	**Bob**
	Key Creation	
	N and e published.	Choose large primes p, q, and compute $N = p \cdot q$. Choose e, with $\gcd(e, (p-1)(q-1)) = 1$.
	Encryption	
Create plaintext m. Use known key (N, e) to compute $c \equiv m^e \pmod{N}$. Send ciphertext c to Bob.	Insecure ciphertext c.	
	Decryption	
		Compute d satisfying $ed \equiv 1 \pmod{(p-1)(q-1)}$. Compute $m' \equiv c^d \pmod{N}$ for plaintext $m = m'$.

Table 1. Table illustrating the steps of the RSA algorithm.

Since Bob chose p and q, only Bob can compute $(p-1)(q-1)$. By the definition of encryption exponent, e is co-prime to $(p-1)(q-1)$, so $\gcd(e, (p-1)(q-1)) = 1$. By Corollary 2.4, Bob can solve

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

and obtain the multiplicative inverse of e modulo $(p-1)(q-1)$, which is the *decryption exponent* d. Using the decryption exponent d, Bob can solve for the plaintext m by computing

$$\begin{aligned} c^d &\equiv (m^e)^d \\ &\equiv m^{de} \\ &\equiv m^{k(p-1)(q-1)+1} \\ &\equiv m \pmod{N}. \end{aligned}$$

The last equality comes from $\phi(N) = (p-1)(q-1)$ and from Euler's Theorem 2.9

4. RSA's security

Whether the RSA public key cryptosystem is secure or not depends greatly on the difficulty of finding the value of $(p-1)(q-1)$. As we illustrated in the previous section§3, if Eve can find the value of $(p-1)(q-1)$, then she will be able to compute decryption exponent d, and decrypt the ciphertext c as Bob does.

Expanding $(p-1)(q-1)$ gives

$$(p-1)(q-1) = pq - p - q + 1 = N - (p+q) + 1.$$

Since Bob published N, N is already known to Eve. Hence now the security of RSA cryptosystem depends on how hard it is to find the sum $p + q$.

Actually, if Eve can find the sum $p+q$, and with the product N already known, she can also find out the value of p and q respectively, by finding the roots of such quadratic formula

$$X^2-(p+q)X+pq.$$

In this case, it means that it is not easier for Eve to find the value of $(p-1)(q-1)$ than for her to find p and q. We can also say that the security of RSA public key cryptosystems relies on the difficulty of this factorization problem.

Problem 4.1 (Integer factorization)**.** Given an integer N promised to be a product of two large primes p and q, find p and q.

However, it does not mean that Eve must factor N in order to decrypt the ciphertext c. Instead of finding p and q, Eve could directly try to solve this congruence

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

where m is the plaintext we try to get, with encryption exponent e, modulus N and the ciphertext c already known to us.

So the fundamental problem of RSA is to solve the congruence.

Problem 4.2 (RSA)**.** Given e, c and N, also with this equation known, find the value of x.

$$x^e \equiv c \pmod{N},$$

In other words, the security of the RSA relied on the assumption that it is hard to compute the e^{th} roots modulo N.

Theorem 4.3. *If the Problem 4.1 is solved, then with the value of p and q known, the product of $(p-1)(q-1)$ can be computed, so that Problem 4.2 can be solved.*

Proof. The operation of solving the RSA problem by using the value of $(p-1)(q-1)$ can be found in §3. □

Solving the factorization problem to get the product $(p-1)(q-1)$ is one way of breaking the RSA problem. However, it is conceivable that there might be other ways to solve for m without factoring N to find the value $(p-1)(q-1)$. But until now, no one has ever found out such a method, and it is unclear if that method might exist.

People have been doing research about the difficulties of these two problems: is breaking the RSA system as hard as factoring integers. Boneh and Venkatesan [1] suggest that, directly solving the congruence, also known as computing the roots modulo N, may be easier than factoring out p and q from N. In other words, Problem 4.2 may be easier to solve than Problem 4.1.

Remark 4.4. With the assumption that the Integer factorization is difficult to solve, RSA is considered very secure and has been widely used for various applications such as key exchange, data transmission and digital signature. However, since the RSA algorithm requires the use of high-digit prime numbers, the computation of message encryption and decryption has slow processing speed, which in some cases, diminish the application of the RSA algorithm. While the quantum computer is developing nowadays, it may help increase the processing speed. But at the same time, people also suggests that quantum computer may be able to actually break the RSA algorithm by also solving the Integer factorization problems discussed above [7].

5. RSA's implementation

5.1. **Finding p and q.** The security of the RSA algorithm hinges on the choice of two large prime numbers, p and q. These primes should be selected randomly and should be of similar but non-identical sizes to protect against certain types of cryptographic attacks (like Fermat's factorization method). Here's how this is typically done:

- Random Number Generation: Start by generating random numbers of the desired bit length. This length is chosen based on the security requirements. For modern encryption standards, each of p and q should be at least 1024 bits long.
- Primality Testing: After a random number is generated, it must be tested to confirm whether it is prime. This is not a trivial task due to the size of the numbers involved.

5.2. **Efficient computation of large exponentiations.** Both encryption and decryption processes in the RSA algorithm require performing exponentiation with very large numbers. Due to the size of these numbers, straightforward methods would be impractically slow. Therefore, RSA implementation typically uses *fast exponentiation* algorithms such as *square-and-multiply* (also known as exponentiation by squaring) to speed up these calculations. This technique significantly reduces the computational complexity from exponential to polynomial time, which is crucial for practical implementations of RSA. For decryption, the method is similar but uses the private exponent d. The decryption formula is computed efficiently using the same square-and-multiply approach, ensuring that even though d is typically a large number, the operation remains computationally feasible.

5.3. **Primality testing.** Primality testing is the process of determining whether a given number is a prime. In the context of RSA, it's important that p and q are prime to ensure the security of the encryption system. Here are the most commonly used methods, which fall into two categories:

5.3.1. *Probabilistic tests.* Probabilistic tests are often favored in cryptographic applications because they offer a good balance between speed and accuracy.

- *Miller-Rabin test*: This is the most widely used probabilistic test in cryptographic applications. It involves checking, under certain conditions, whether a number can be written in a form that identifies it as a composite (non-prime). A number passes the test several times with different random bases to increase the certainty of its primality. Although there is a small chance that a composite number might pass the test (false positive), this probability decreases exponentially with more rounds of testing.
- *Solovay-Strassen test*: This test is similar to Miller-Rabin but uses Euler's criterion to check for primality. It is less popular than Miller-Rabin due to certain inefficiencies and similar error probabilities.

These tests typically do not prove definitively that a number is prime but can say with high probability that it is not composite. For example, after 20 rounds of Miller-Rabin, the probability that a composite number is mistakenly considered prime is less than 2^{-20}, or about 1 in a million.

5.3.2. *Deterministic tests.* When absolute certainty is required, deterministic tests can be used, though they are generally slower and less practical for large numbers typical of RSA.

- *AKS* Primality test: The first primality test proven to be both general (works for all numbers) and polynomial in time complexity. However, it is rarely used in practice because simpler probabilistic tests are much faster and provide sufficient certainty for cryptographic purposes.
- *Elliptic curve primality proving (ECPP)*: This method can provide a certificate of primality and is often faster in practice than AKS, although its theoretical basis is more complex. It's used when a non-probabilistic proof is needed for smaller primes.

5.4. **Implementation in RSA.**

When implementing RSA, primes p and q are typically generated using a cryptographic library that includes efficient implementations of these tests. These libraries usually employ a combination of a simple test for divisibility by small primes to quickly eliminate obvious non-primes, and several rounds of a probabilistic test like Miller-Rabin for larger numbers.

Given the importance of choosing strong, secure primes, most implementations also adhere to specific guidelines provided by cryptographic standards, such as those from NIST, which recommend certain sizes for the primes and specific parameters for the tests to minimize the risk of choosing weak primes.

5.5. **ASCII conversion of messages.**

Before encryption, plaintext messages need to be converted into numbers, because the RSA algorithm works with numbers mathematically. One common method to achieve this is by using ASCII values, [8]:

- *ASCII encoding*: Each character in the plaintext is replaced with its corresponding ASCII code. For example, the letter 'A' corresponds to 65, 'B' to 66, and so on.
- *Message conversion*: Once each character of the message is converted to ASCII, these numbers are typically concatenated or otherwise combined into a larger number that will be the actual input to the RSA encryption formula, as long as this number is less than $n = p \times q$.

5.6. **Implementation example.**

We completed an implementation based on this example[3]. The code can be found here.

Acknowledgments

We would like to express our sincere gratitude to our mentor, Honglin Zhu, for his expert guidance and invaluable support throughout our study of cryptography. His insightful ideas and feedback were crucial in determining the direction of our paper and the content of our presentation.

We would also like to thank Marisa Gaetz and Mary Stelow for coordinating the MIT PRIMES Circle, giving us the opportunity to embrace the beauty of cryptography, and providing the necessary resources throughout this semester. Special thanks to Marisa for her constructive suggestions on our paper and presentation.

References

[1] Dan Boneh and Ramarathnam Venkatesan, *Breaking RSA may not be equivalent to factoring*, Advances in Cryptology — EUROCRYPT'98 (Berlin, Heidelberg) (Kaisa Nyberg, ed.), Springer Berlin Heidelberg, 1998, pp. 59–71.

[2] Whitfiled Diffie and Martin Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), 644–654.

[3] Geeksforgeeks, *RSA algorithm in cryptography*, Available online, 2023, Retrieved from https://www.geeksforgeeks.org/rsa-algorithm-cryptography/.

[4] Jeffrey Hoffstein, *Discrete logarithms and Diffie-Hellman*, pp. 59–112, Springer New York, 2008.

[5] Jeffrey Hoffstein, *Integer factorization and RSA*, pp. 113–189, Springer New York, 2008.

[6] Jeffrey Hoffstein, *An introduction to cryptography*, pp. 1–58, Springer New York, 2008.

[7] Moolchand Sharma, Vikas Choudhary, RS Bhatia, Sahil Malik, Anshuman Raina, and Harshit Khandelwal, *Leveraging the power of quantum computing for breaking rsa encryption*, Cyber-Physical Systems **7** (2021), 73–92.

[8] The Unicode Consortium, *The unicode standard*, Mountain View, CA, 2021, Retrieved from https://www.unicode.org/versions/Unicode14.0.0/.

Cambridge Rindge and Latin
Email address: `joshua.pite@gmail.com`

Boston Green Academy
Email address: `yyzhong903@gmail.com`