RSA Encryption and Modular Arithmetic

Mack Gregory Yuan Chang Ryan Huang

November 22, 2021

Abstract

RSA encryption was one of the first asymmetric cryptosystems. It is based on modular arithmetic with prime numbers. It's security is believed to be in the fact that we currently do not know of any polynomial time algorithms that factor numbers without the use of a general purpose quantum computer. Rivest, Shamir, and Adleman created the algorithm based off of the work of Diffie and Hellman which proposed a way that a symmetric key could be publicly created. An equivalent system to RSA years was made by Cocks while working for GCHQ, however it remained classified for many years, leaving Rivest, Shamir, and Adleman to take credit for its creation. RSA can be broken into three different related processes key generation; encryption and decryption; signing and signature verification.

1 Historical Background

Encryption of messages is not a new task. The task of secretly transmitting messages securely has been important to parties for thousands of years. A common thread to all of the methods up until the RSA algorithm is that they used symmetric encryption, that is the same key that encrypts a given message can also decrypt the message. One exceptionally weak example of a symmetric encryption would be the Caesar Cipher used by Julius Caesar, which was one of the first known uses of cryptography of any kind [1, pg 311]. In this cipher we pick

a key k between 1 and 25. We then encrypt each letter of the message is by replacing it with the letter plus k, wrapping around to the beginning of the alphabet if necessary. Thus if we let our $k = 3$ and message be "hello world", then we would sent the ciphered text "khoor zruog". The decryption process works the same way, but instead, we subtract k from each letter of the ciphertext, wrapping around to the end of the alphabet if necessary. Of course with modern technology we can have a computer try all 25 possible values of k until we decide which value makes a coherent message. A problem with symmetric cryptography like a caesar cipher is that they must know the key ahead of time, and it shouldn't be sent insecurely as anyone with the key can read and send messages that appear to be from the trusted other party.

Diffie and Hellman created an algorithm that allowed for a symmetric key to be created over insecure communications. The Diffie Hellman key exchange algorithm starts with the two parties i and j agreeing upon values for q and α, such that α is a relatively small number. q should be chosen so that it is around 2048 to 4096 bits long to prevent easy breaking of the key [2]. Each party has these numbers, but so do any attackers. Each party then generates a random number X_i and X_j such that $1 < X < q - 1$. They each then compute $Y = \alpha^X \mod q$ and transmit it to each other. They finally generate the symmetric key for their communications by $K_{ij} = {Y_i}^{X_j} \mod q = {Y_j}^{X_i} \mod q = \alpha^{X_i \cdot X_j} \mod q$ [3]. Any attackers may have been able to intercept q, α, Y_i, and Y_j however these values do not give them enough information to easily compute the newly created key, K_{ij}.

To give an example of this, we let $\alpha = 11$, $q = 31$. We also choose $X_i = 7$ and $X_j = 4$. We compute $Y_i = \alpha^{X_i} \mod q = 11^7 \mod 31 = 13$, and $Y_j = \alpha^{X_j} \mod q = 11^4 \mod 31 = 9$. We exchange our Y values and find that our symmetric key would be ${Y_i}^{X_j} \mod q = 13^4 \mod 31 = 10 = {Y_j}^{X_i} \mod q = 9^7 \mod 31 = 10$. Through these set of numbers, now both the i and j party can communicate using a symmetric key encryption algorithm of their choice.

2 Mathematical Background

2.1 Useful Definitions

An integer p is **prime** if it is divisible by only ± 1 and $\pm n$ [4, pg. 62]. Some examples of primes include 2, 3 and 5.

An integer is **composite** if it is ≥ 2 and is not prime [4, pg. 62].

We define **coprime** as a binary relation that means that the greatest common divisor of two numbers is equal to 1. This may also be called relatively prime.

We define the function called **modulo** or mod be defined for any $m \in \mathbb{Z}$ and $n \in \mathbb{N}$. $m \mod n = r$ is such that $0 \leq r \leq n-1$ and there exists $q \in \mathbb{Z}$ such that $m = q \cdot n + r$ [1, pg. 253]. For example $23 \mod 5 = 3$ as there exists $q = 4$ such that $23 = 4 \cdot 5 + 3$ and $0 \leq 3 \leq 5 - 1$.

We define the relation **congruent** as $\equiv$ on $\mathbb{Z}$ as for all $x, y \in \mathbb{Z}$ and $n \in \mathbb{N}$ such that $x \equiv y(\mod n) \iff n \mid x - y$. ie $8 \equiv 3(\mod 5)$ as $8 - 3 = 1 \cdot 5$.

We define Big-O or $\mathcal{O}$ as a way to classify the output of functions that typically represent the running time or memory occupation of algorithms into sets. Let $f(n)$ and $g(n)$ be functions from $\mathbb{N}$ to $\mathbb{N}$. We can say that $f(n) \in \mathcal{O}(g(n))$ if and only if there exists $c, n_0 \in \mathbb{N}$ such that $\forall n > n_0, f(n) < c \cdot g(n)$ [5, pg. 47]. We can also say that $f(n)$ is asymptotically upper bounded by $g(n)$. For example let $f(n) = 5n^2 + 3n + 5$ and $g(n) = n^2$. We aim to show that $5n^2 + 3n + 5 \in \mathcal{O}(n^2)$. Let $n_0 = 1$ and $c = 13$. We know that $5n^2 \leq 5n^2$, $3n \leq 3n^2$, and $5 \leq 5n^2$. Therefore we know that $5n^2 + 3n + 5 \in \mathcal{O}(n^2)$. We define the set P, as the set of all problem statements for which an algorithm exists whose operations can be modeled by$f(n) \in \mathcal{O}(n^k)$ for some fixed $k \in (\mathbb{N} \bigcup \{0\})$

We define the set Z_k to be the set of all integers n such that $a \leq n \leq k$.

We define a **cryptosystem** as having a set of plaintext strings, ciphertext strings, keys (called the keyspace), encryption functions, and decryption functions [1, pg. 314]. We can say that the Caesar cipher is a cryptosystem as the set of plaintext is given by $\mathbb{Z}_{26}$,

the set of ciphertext is given by $\mathbb{Z}_{26}$, the set of possible keys is $\mathbb{Z}_{26}$, the set of encryption functions is $\{\forall k \in \mathbb{Z}_{26} | E_k(n) = n + k \mod 26\}$, and the set of decryption functions is $\{\forall k \in \mathbb{Z}_{26} | D_k(n) = n - k \mod 26\}$.

A **private key cryptosystem** is one that has a decryption key that can be easily known from the decryption key. In the case of the ceasar cipher this is instant as the keys are identical. A **public key cryptosystem** comparitvetely doesn't allow one to easily know an decryption key from an encryption key.

We define Euler's Totient Function ϕ as a function that measures the amount of numbers less than the input that are coprime to the input. It is defined on natural numbers n in the following way $\phi(n) := |j \in \mathbb{N} : \gcd(n, j) = 1|$ [6].

2.2 Modular Arithmetic Background

In most cases computers store discrete values that occupy a fixed size in a computers memory. For modern computer hardware the fixed size is 64 bit, which means that the central processing unit and the arithmetic logic unit inside of it can only handle a number 64 bits long. Unfortunately for integers that means that we can only represent integers from 0 to $2^{64} - 1$. If one tried to represent an integer 2^{64} or larger on that machine, software could be written that would operate on the bits, but it doesn't operate nearly as fast as letting your computer's hardware do it instead. When we look at modern cryptography we know that a 64 bit number would be foolish to use for the security of your data. If we look at the certificate for `https://www.ucdavis.edu` we see that it uses a 2048 bit modulus in its RSA encryption. However we can be assured that the computer is not directly working with numbers that large in its hardware. But what tricks does it use to speed up the math? Modular arithmetic used in several cryptographic schemes help us to keep our computation sizes from reaching numbers that the hardware cannot handle.

Proposition M.1 $\forall a, b \in Z, n \in \mathbb{N}, ((a \mod n) + (b \mod n)) \equiv (a + b) (\mod n)$

This is true if and only if we can show that n divides $a+b-((a \mod n)+(b \mod n))$ by definition of congruent.

Let p be the integer equal to $a+b-((a \mod n)+(b \mod n))$ By the division algorithm, there exists $j,k \in \mathbb{Z}, r,s \in \mathbb{N}$ such that $a = jn+r$ and $b = kn+s$

By replacement $p = jn+r+kn+s-((a \mod n)+(b \mod n))$

By definition of subtraction $p = jn+r+kn+s+-1((a \mod n)+(b \mod n))$

By distributivity $p = jn+r+kn+s+-(a \mod n)+-(b \mod n))$

By definition of mod, $a \mod n = r$ and $b \mod n = s$

By replacement $p = jn+r+kn+s+-r+-s$

By distributivity $p = n(j+k)+r+s+-r+-s$

By additive inverse $p = n(j+k)$

By definition of divisibility n divides p. By replacement n divides $a+b-((a \mod n)+(b \mod n))$ $\square$

Proposition M.2 $\forall a,b \in Z, n \in \mathbb{N}, ((a \mod n)\cdot(b \mod n)) \equiv (a\cdot b)(\mod n)$

This is true if and only if $a\cdot b-((a \mod n)\cdot(b \mod n))$ is divisible by n by definition of congruent

Let p be the integer equal to $a\cdot b-((a \mod n)\cdot(b \mod n))$

By the division algorithm there exists an integer j such that $a = jn+a \mod n$ and integer k such that $b = kn+b \mod n$

By replacement $p = (jn+a \mod n)\cdot(kn+b \mod n)-((a \mod n)\cdot(b \mod n))$

By distributivity $p = jn(kn+b \mod n)+(a \mod n)(kn+b \mod n)-((a \mod n)\cdot(b \mod n))$

By commutativity of multiplication $p = jn(kn+b \mod n)+(kn+b \mod n)(a \mod n)-((a \mod n)\cdot(b \mod n))$

By distributivity $p = jn(kn+b \mod n)+kn(a \mod n)+(b \mod n)(a \mod n)-((a \mod n)\cdot(b \mod n))$

By commutativity of multiplication $p = jn(kn+b \mod n)+kn(a \mod n)+(a \mod n)(b$

$\mod n) - ((a \mod n) \cdot (b \mod n))$

By definition of subtraction $p = jn(kn+b \mod n)+kn(a \mod n)+(a \mod n)(b \mod n)+ -((a \mod n) \cdot (b \mod n))$

By additive inverse $p = jn(kn + b \mod n) + kn(a \mod n)$

By commutativity of multiplication $p = n(j(kn + b \mod n)) + n(k(a \mod n))$

By distributivity $p = n((j(kn + b \mod n)) + (k(a \mod n)))$

By definition of divisibility n divides p. We also know that n divides $a \cdot b - ((a \mod n) \cdot (b \mod n))$. Therefore for all $a, b \in \mathbb{Z}$ and $n \in \mathbb{N}$, $((a \mod n) \cdot (b \mod n)) \equiv (a \cdot b)(\mod n)$ $\square$

Binary Modular Exponentiation Algorithm [7]

Let b be a natural number which we exponentiate. Let p be the natural number power which we exponentiate b Let n be the natural number modulus. We wish to find $b^p \mod n$ We define a function $f(c, d) :=$ if $c = 1$ then d otherwise 1. c must be an integer, and d must be an integer. As such the function maps from $(\mathbb{Z} \times \mathbb{Z}) \to (\mathbb{Z})$ We choose the smallest natural number i such that $2^i \geq p$. Let j be a natural number equal to this i. Let q be the variable natural number that starts as being equal to p. We make a sequence A such that if $p \mod 2^i = q$, then $A_{j-i+1} = 0$ otherwise $A_{j-i+1} = 0$. We then recursively define future items of the sequence by replacing q with $q \mod 2^i$ and i with $i - 1$ until $i - 1$ is 0.
In this way $p = \sum_{k=0}^{j-1} A_k \cdot 2^k$. We then construct an integer v such that $v = (\prod_{k=0}^{j-1} (f(a_k, b^{2^k}))$ $\mod n) \mod n$ v will be equal to $b^p \mod n$, so we return this from the algorithm.
This makes a useful mathematical definition of the algorithm, although, on a computer, we note that n is noted as a binary number already from which we can extract any bit in $\mathcal{O}(1)$ time. With that, we already have A by virtue of having p already as a binary number. Additionally, we will save some computation by starting at the least significant bits and working our way towards the most significant bit, while storing the previous value of $b^{2^i} \mod n$, as the next iteration can use $(b^{2^i} \mod n)^2 \mod n$ then instead of $(b^{2^{i+1}} \mod n) \mod n$. Additionally, we note that "more efficient procedures" exist that allow encryption

and decryption in approximately half the time [8, pg 8].

Fundamental Theorem of Arithmetic

For all integers $n \geq 2$, there exists a unique and finite sequence of prime numbers $(p_i)_{i=0}^{k} \in \mathbb{Z}$ and a sequence of natural number powers $(a_i)_{i=0}^{k} \in \mathbb{N}$ such that $n = \prod_{j=0}^{k} (p_j^{a_j})$. The sequences are unique for a given n except for possible ordering changes [9]. If $n \in \mathbb{Z}$such that$n \geq 2$, then n can be uniquely factored into a product of primes numbers. If there are two different prime factorization of the number n, then every prime number in each factorization must occurs the same number of times, thus the difference is only the ordering.

Fermat's Little Theorem

Let p be an integer such that p is prime, and $a \in \mathbb{N}$ such that is not divisible by p.

$a^{p-1} \equiv 1 (\text{mod } p)$ is true by Fermat's Little Theorem.

While this theorem is interesting and can save a lot of computational cost and overflow, it doesn't typcally serve much purpose towards RSA as the modulo used will never be a prime number.

Chinese Remainder Theorem[10]

Let a set of r integers n be coprime. Given a system of r congruences

$$\begin{aligned} x &\equiv b_1 \mod n_1 \\ x &\equiv b_2 \mod n_2 \\ &\vdots \\ x &\equiv b_r \mod n_r, \end{aligned}$$

we can create a solution x that is unique mod $\prod_{i=0}^{r} n_i$ This solution allows us to break a product modulus up into factors to reduce the complexity of the problem in some circumstances. Consider if we were only talking about two congruence, then it would be easier to

compute the value with this theorem since by making the system from the congruence based on the product of n_1 and n_2, we have ensured that the n's are prime and will be able to use Fermat's little theorem as well.

3 RSA Algorithm

3.1 Key Generation

RSA key generation starts with acquiring two approximately equally large prime numbers that we will call p and q. Typically these numbers are found through a random number generator and the a primality test. We multiply p and q to make n, the modulus used in both our public and private keys. $\phi(n)$ is computed as the product of $p-1$ and $q-1$ as the are both primes under Euler's totent function. We also choose e such that e is co-prime to $\phi(n)$ as well as less than it. We now must calculate d such that $e \cdot d \cong 1(\mod \phi(n))$. d is also called the modular inverse of e.

Now that we have created all parts of our digital key, we need to consider which parts can be freely shared, which parts need to be hidden in a safe location, and which parts are no longer necessary. n and e make the public key that is shared with everyone that you would like to send you an encrypted message, and it will not compromise any of your security if you send your public key to any attackers. The private key n and d should be safely stored, as anyone with the private key can impersonate you digitally and read encrypted communications meant for your eyes only. Finally you should no longer hold onto p, q, and phi. These could be used to recreate your keys if someone got a hold of them, yet they have no use for you anymore.

3.2 Encryption and Decyption

When someone wants to encrypt a message, they will make it a number we call m. m must equal $m \mod n$ or else it will not be able to be correctly reproduced. This means that m

cannot be negative, and it cannot be greater than or equal to n. If the message satisfies those constraints however, we then compute the ciphertext $c = m^e \mod n$. To retrieve the message the reciepient must calculate the message as $m = c^d \mod n$. We see here how some of the constraints come into play, as $m = (m^e \mod n)^d \mod n = m^e d \mod n = m^1 \mod n = m \mod n = m$.

We note that all messages and ciphertext is in the set $\mathbb{Z}_n$ as the values of each must be equal to themselves $\mod n$.

We now define two functions $E_{(n,e)}(m) := m^e \mod n$ and $D_{(n,d)}(c) := c^d \mod n$, which encrypt and decrypt given the appropriate key

We note that $m = D_{(n,d)}(E_{(n,e)}(m))$ and $c = E_{(n,e)}(D_{(n,d)}(c))$ which means that the two functions act as inverses of each other, and that there is a one-to-one mapping between the message space and the ciphertext space.

3.3 Signing messages

Just as the algorithm works to encrypt and decrypt messages, it also works to prove that someone has sent a message. The sender creates their message w such that $w = w \mod n$. They then use their private key to encrypt w into a cipertext t, such that $t = w^d \mod n$. They then can ensure that only a certain recipent can read their signed message t by further encrypting the message with the intended recipients A's public key pair (n_A, e_A). In this way the sender now will only send $(w^d \mod n)^{e_A} \mod n_A$ which will ensure that only the intended recipient A will be able to open the message, they can be certain that the message came from the sender as $w = t^e \mod n$, and they will not be able to make a forge the senders signature on a different message as they cannot compute $\hat{w}^d \mod n$.

resulting in send both w and t. The sender is the owner of the private key, and the reciepient will use the public key to retrieve the hash, and verify who sent the message and the integrity of it by calculating $h = t^e \mod n$ and checking it against their independently

computed hash of w. Just like encryption the hash must be encrypyted as a number, and the number must be congruent with itself modulo n.

4 Modern Security Implications of RSA

Several successful attacks have occured against including a poor implementation, small values of e, and small values of n. Each of these ends up recieving wide media attention and disrupting peoples faith in the security used in many of their electronic communications.

4.1 Poor Implementation of RSA

Often to generate prime numbers p and q, a random number is chosen through a psuedo-random number generator, and then a primality test is used to find a prime numbers around that size. This has lead to the possibility of attacks in some devices that are not able to generate random numbers well. As a consequence, attackers can freely find keys all over the internet that share common factors, allowing them to be factored through the use of an greatest common factor algorithm [11, pg 29]. In large part this flaw can be attributed to dependance upon user interation for randomness (such as mouse movement or microphone input) in some computer operating systems or a lack of a hardware randomness implementation.

4.2 Attacks using small modulus

As a contest many numbers have been proposed as challenges to benchmark how long it might take for an attacker with lots of resources to factor values of n from public keys so that d can be recomputed. Through these contests, several large numbers have been factored into two prime factors, including RSA-129 in 1994, and RSA-768 in 2009 [10]. The largest

of the numbers that has been factored so far is known as RSA-240 [12].

$$
\begin{aligned}
&124620366781718784065835044608106590434820374651678805754818788883289666801188210855036039570272508747509864768438458621054865537970253930571891217684318286362846948405301614416430468066875699415246993185704183030512549594371372159029236099 = \\
&509435952285839914555051023580843714132648382024111473186660296521821206469746700620316443478873837606252372049619334517 * \\
&244624208838318150567813139024002896653802092578931401452041221336558477095178155258218897735030590669041302045908071447
\end{aligned}
$$

The computational resources put into factoring RSA-240 would not be all that different that what attackers might use against the encryption of a bank, software distributor, or otherwise valuable target. As such, we know that it would not be reasonable to continue using numbers as small as RSA-240 into the future. A 1024-bit RSA modulus is not uncommon for online communications, however it would be prudent to stay ahead of the curve and begin to use 2048-bit RSA moduli as older certificates expire.

4.3 Shor's Algorithm

Out of all of these threats to RSA's security, quantum computing could soon become the most dangerous. Shor's algorithm promises the ability to place the integer factorization problem in $\mathcal{P}$ for a quantum computer [13, pg. 131]. This algorithm would allow for a much quicker factorization of n into p and q, where d could then be computed allowing for all messages encrypted using that key to be read. Luckily for now, a quantum computer hasn't been able to successfully factor any number larger than 21, which is only 5 bits long. If researchers are successful in creating quantum computers into the future, it could greatly ruin RSA's security.

A recent development in this quantum threat to our security has been the revealing of a quantum computer that claims to have twice the quantum volume (a metric that claims to measure the ability of quantum computers to solve complex problems) of competitors, with predictions of exponential growth to quantum computers abilities going forward in a way that resembles Moore's law [14].

5 Conclusion

The RSA algorithm comprises much of the modern internet infrastructure, however several threats to its strength are starting to appear that could seriously threaten its security into the future. As such many uses of RSA should be switched over to use at least 2048-bit moduli or potentially even to a newer algorithm to provide secrecy into the future.

References

[1] Kenneth H. Rosen. *Discrete Mathematics and Its Applications*. 8th ed. McGraw-Hill Education, 2019. ISBN: 978-1-259-67651-2.

[2] Micheal Pound and Sean Riley. *Diffie Hellman -the Mathematics bit*. Computerphile. Dec. 20, 2017. URL: https://www.youtube.com/watch?v=Yjrfm_oRO0w.

[3] W. Diffie and M. Hellman. "New Directions in Cryptography". In: *IEEE Transactions on Information Theory* (Nov. 1976). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1055638&tag=1.

[4] Matthias Beck and Ross Geoghegan. *The Art of Proof*. Ed. by S. Axler K.A. Ribet. Springer, 2010. ISBN: 978-1-4419-7022-0.

[5] Thomas H. Cormen et al. *Introduction to Algorithms*. 3rd ed. The MIT Press, 2009. ISBN: 978-0-262-03384-8.

[6] Yuan Chang et al. "RSA Encryption and Modular Arithmetic". Contact mgr@ucdavis.edu for access to repository. Feb. 19, 2020. URL: https://github.com/sac-bsa/RSA_term_paper/commit/ae46097038057674ab13aa55888766a3fe736b76#diff-23e598184b3ae2495cdb947f8d9516ec.

[7] Colin Godfrey. *Binary Modular Exponentiation*. Sept. 22, 2015. URL: https://www.cs.umb.edu/~cgodfrey/handouts/CS320-F2015-9-22supplement.pdf.

[8] R.L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signitures and Public-key Cryptosystems". In: *Association for Computing Machinery* (Feb. 1978). URL: https://people.csail.mit.edu/rivest/Rsapaper.pdf.

[9] Bruce Ikenaga. *The Fundamental Theorem of Arithmetic*. June 14, 2008. URL: https://www.math.uh.edu/~minru/spring11/fundamental-theorem.pdf.

[10] Ryan Huang et al. "MAT 108 Term Paper Draft". Contact mgr@ucdavis.edu for access to repository. Feb. 23, 2020. URL: https://github.com/sac-bsa/RSA_term_paper/commit/72876fd1992320292aaab8e6dd14631da682b045#diff-1fcdb9f364e205040d115073355abb

[11] Nadia Haninger. *How Not to Generate Random Numbers*. University of Pennsylvania. May 13, 2015. URL: http://web.stanford.edu/class/ee380/Abstracts/150513-slides.pdf.

[12] Emmanuel Thomé. *795-bit factoring and discrete logarithms*. Institut national de recherche en sciences et technologies du numérique. Dec. 2, 2019. URL: https://lists.gforge.inria.fr/pipermail/cado-nfs-discuss/2019-December/001139.html.

[13] Peter W. Shor, ed. *Algorithms for Quantum Computation: Discrete Logarithms and Factoring*. Proceedings 35th Annual Symposium on Foundations of Computer Science (Santa Fe, NM, USA). 1994.

[14] Paul Smith-Goodson. *Honeywell Subprisingly Announces It Will Be Releasing The Most Powerful Quantum Computer In The World*. Mar. 3, 2020. URL: https://www.forbes.com/sites/moorinsights/2020/03/03/honeywell-surprisingly-announces-it-will-be-releasing-the-most-powerful-quantum-computer-in-the-world/#352a4d7614b4.

Further Reading

Henry Reich. *How Quantum Computers Break Encryption — Shor's Algorithm Explained.*

URL: https://www.youtube.com/watch?v=lvTqbM5Dq4Q