IB Math SL Yr. 2 Internal Assessment

An Exploration in Modular Arithmetic through RSA Encryption

Table of Contents

Introduction

In World War II, there were many important messages sent between parties, both in the Allies and the Axis Powers. These messages had to be encrypted because they often contained sensitive information and they were often intercepted by the enemy. To do so, the German army developed a machine called the Enigma. The Enigma was a machine that used symmetric encryption (encryption in which the sending party and the receiving party have the same key) to send encrypted messages. This machine frustrated the Allies for quite a while. However, due to the necessity of sharing the private key, the encryption was eventually broken (Carlson).

In today's world, there is another war being fought. This is a war over data. There are many malicious parties that want the information that is sent through the internet. The difference between this war and World War II is that technology has advanced, therefore increasing the processing power of the computers that the enemy has. This makes it a lot easier to break encrypted messages. It is therefore important to develop strong encryption methods.

The Diffie-Hellman Exchange

In the 1970s, Whitfield Diffie, a former student at MIT, realized the potential for what was going to become the Internet, as well as the need to protect information sent across this network. Together with Martin Hellman, a Stanford student who studied key-distribution, and Ralph Merkle, a computer scientist, they created a concept that would change the encryption community forever: the Diffie-Hellman exchange.

The idea behind the Diffie-Hellman exchange is that the plaintext t is encrypted with a function $f(x)$. It should be very simple to calculate the cipher text $f(t)$ when both t and $f(x)$ is known; however, when only the function $f(x)$ and the resulting cipher text $f(t)$ are known, it should be impossible for all practical purposes to discover the plaintext t. In other words, the inverse function $f^{-1}(x)$ should be nearly impossible to calculate.

Diffie and Hellman searched hard for a function that fit the requirements they had set forth. However, after a year of searching, they turned up empty handed and decided instead to publish the idea in a paper and leave the discovery of the function to someone else. (Jankvist 158-159)

The Invention of RSA

Two other students from MIT, Ronald Rivest and Adi Shamir, got a hold of the paper published by Diffie and Hellman. They began to work on an equation that would satisfy the requirements set forth by Diffie and Hellman. They spent time coming up with solution after solution only to have it shot down by their friend and colleague, Leonard Adleman. Finally, after having 42 solutions broken by Adleman, Rivest decided to propose a solution that uses the difficulty of factoring large numbers, as well as the difficulty of finding the inverse of functions that incorporate modular arithmetic. This satisfied the requirement for a function, multiplying numbers together, that is easy to do, but undoing it, or prime factoring, was extremely challenging. This solution was unable to be broken by Adleman and therefore accepted by the group as a strong encryption method.

Rationale

In 10th grade, I participated in a "Capture The Flag" competition called *picoCTF* at the prompting of an instructor. At the time, I didn't know what it was about. As I participated, I learned that *picoCTF* is a hacking competition in which they give you a set of challenges. In each challenge, you are supposed to use a Linux terminal to accomplish the task assigned and find the flag. One challenge involved RSA encryption. To complete the task, I simply found a website that decrypted the message. However, I spent a little bit of time looking up RSA and noticed that it involved something called "modular arithmetic". However, I did not have time then to research very far and forgot about it. Later, when I was considering my topic for my Math IA, I was reminded of RSA encryption. I decided to use RSA encryption for my Math IA topic so that I would have an excuse to find out more about RSA encryption.

Aim

The purpose of this Math IA is to inform the reader about how modular arithmetic is used in the process of RSA encryption. I also am writing this paper to ensure that I know this process thoroughly and understand the reasoning behind each step so that I can use it later if I decide to go into a career in digital security, which is one career that I am looking at.

Exploration

In my exploration, I am going to take the reader through the process of RSA Encryption. First, I will give the reader a basic explanation of modular arithmetic. Next, I will describe the process of generating the RSA key and the importance of the properties of the RSA key. Finally, I will describe the encryption and decryption process.

Basic Modular Arithmetic

Modular arithmetic is also commonly known as “clockwork arithmetic”. This is because it works in the same way that a clock does. A clock has 12 hours represented. The clock starts at 12, then it goes to 1. Next, it increments by 1 until it arrives at 12 again. Then it starts over at 1. This is shown in Figure 1.

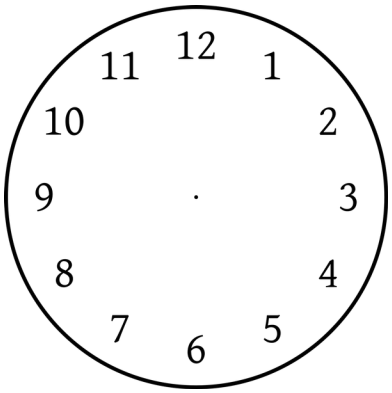

Figure 1 | A clock goes from 1 to 12 before starting over at 1 again

Modular arithmetic is similar. However, it starts at 0 and returns to 0 when the value reaches what is called the modulus. In the case of the clock, the modulus would be 13 because the clock resets when the value reaches 13.

Basic Form

The basic idea behind modular arithmetic is that every value v can be represented modulus m by

$$v \in \{mk + 0, mk + 1, mk + 2, \ldots mk + (m - 1)\}$$

where k is some constant integer. Modular arithmetic simply ignores any number multiplied by m, making

$$v \pmod m \in \{0, 1, 2, \dots m - 1\}$$

Modular arithmetic in it's most basic form is represented in mathematics in the following form, called an equivalence relation or congruence relation:

$$a \equiv b \pmod m$$

where b and m are positive integers and a is any integer. This is read as *a is congruent to b modulo m*.

If value $v \notin \{0, 1, 2, 3, \dots m\text{-}1\}$, v modulus m will be the remainder of v / m which can be found using

$$v \pmod m = (v / m - \lfloor v / m \rfloor) \times m$$

Example: Basic Modular Arithmetic

let $m = 6$ *and* $v = 21$

$21 \notin \{0, 1, 2, 3, 4, 5\} \therefore 21 \pmod 6 = (21 / 6 - \lfloor 21 / 6 \rfloor) \times 6$

$21 \pmod 6 \equiv 3$

Note that the above equation works for both positive and negative values. However, if $v < 0$ and $|v| \leq m$, a simple way to find $v \pmod m$ is as follows:

$$v \pmod m \equiv m + v$$

Example: Negative Values

let $m = 6$ *and* $v = -4$

$-4 \pmod 6 \equiv 6 + (-4) \equiv 2$

Addition in Modular Arithmetic

Addition in modular arithmetic can be done in two ways. The first is by adding up all values, then finding the sum modulus m. However, another way that keeps the sum

small is by finding each number modulus m before adding them up. This can be done because of how modular arithmetic works. In a modular addition problem

$$a + b \equiv c \pmod{m}$$

we can rewrite it to say that

$$(mk_a + r_a) + (mk_b + r_b) \equiv c \pmod{m}$$

However, mod m means that mk_a and mk_b can be ignored. This therefore makes the equation

$$(mk_a + r_a) + (mk_b + r_b) \equiv r_a + r_b \pmod{m}$$

Example: Addition

let $a = 12$, $b = 14$, and $m = 5$

$12 + 14 = 26$

$12 \equiv 2 \pmod{5}$

$14 \equiv 4 \pmod{5}$

$26 \equiv 1 \pmod{5}$

$12 + 14 \equiv 2 + 4 \equiv 6 \equiv 1 \pmod{5}$

Multiplication in Modular Arithmetic

Multiplication in modular arithmetic can also be done by standard multiplication, or it can be simplified by finding each number modulus m before multiplying. This is also because of the idea behind modular arithmetic. Given a modular multiplication problem $ab \equiv c \pmod{m}$, we can rewrite it to say that

$$(mk_a + r_a)(mk_b + r_b) \equiv c \pmod{m}$$

$(mk_a + r_a)(mk_b + r_b)$ can be distributed to $mk_amk_b + mk_ar_b + mk_br_a + r_ar_b$. Because we can ignore anything multiplied by m,

$$mk_amk_b + mk_ar_b + mk_br_a + r_ar_b \equiv r_ar_b \pmod{m}$$

Example: Multiplication

let a = 12, b = 7, and m = 5

$12 \times 7 = 84$

$12 \equiv 2 \pmod 5$

$7 \equiv 2 \pmod 5$

$84 \equiv 4 \pmod 5$

$12 \times 7 \equiv 2 \times 2 \equiv 4 \pmod 5$

RSA Key Generation

In RSA key generation, the goal is to end up with three numbers: *N*, *e*, and *d*. *e* is the value that is key to the encryption function. *d* is the value that is key to the decryption function. *N* is the modulus to both of these functions. The functions are given in the section *RSA Encryption and Decryption*. These keys are generated by following these steps. I will also provide an example along with the steps that builds an RSA key set. However, the examples will use relatively small numbers and are therefore only for demonstration. When implemented in real-world applications, the keys are recommended to be 2048 bits to 3072 bits, which is anywhere from 615 digits to 925 digits long (United States 35).

Find *N*

The first step in the process is to find *N*. *N* is found simply by multiplying two predetermined primes, *p* and *q*. *N* is going to be used as the modulus for encryption and decryption. *p* and *q* must be primes for two reasons. The first is that it makes the usage of Euler's Totient Theorem (which is explained and used in the section titled *Finding e*) much simpler. The second is that it makes the RSA encryption method harder to break, as it makes *N* harder to factor.

***Example: Finding* N**

let p = 17 and q = 31

N = p × q = 17 × 31 = 527

Find *e*

The next step in the process is to find *e*. To find *e,* it is important to know the definition of coprime. If two numbers are coprime, this means that they share a greatest common divisor (gcd) of 1.

Example: Coprime

gcd(21, 22) = 1 ∴ 21 and 22 are coprimes

gcd(21, 24) = 3 ∴ 21 and 24 are not coprimes

To find *e*, it is also necessary to use Euler's Totient Theorem. Euler's Totient Theorem defines a totient function, $\phi(n)$, that gives the number of values that are less than *n* and that are coprime to *n*. Euler's Totient Theorem states that for any prime *p*,

$$\phi(p) = p - 1$$

Euler's totient function is also multiplicative which means that

$$\phi(a) \times \phi(b) = \phi(ab)$$ (Cruise *Euler's Totient Theorem*).

Therefore, because $N = p \times q$ and *p* and *q* are prime, $\phi(N) = \phi(p) \times \phi(q) = (p-1)(q-1)$.

Example: Euler's Totient

let a = 5 and b = 7

let A be all numbers between 0 and 5 that are coprime to 5

A = {1, 2, 3, 4}

let B be all numbers between 0 and 7 that are coprime to 7

B = {1, 2, 3, 4, 5, 6}

ϕ(5) = length of A = (5 - 1) = 4

ϕ(7) = length of B = (7 - 1) = 6

let c = ab and C be all numbers between 0 and c that are coprime to c

c = 5 × 7 = 35

C = {1, 2, 3, 4, 6, 8, 9, 11, 12, 13, 16, 17, 18, 19, 22, 23, 24, 26, 27, 29, 31, 32, 33, 34}

ϕ(35) = length of C = (5 - 1)(7 - 1) = 4 × 6 = 24

Now that we know *ϕ(N),* we use it to determine *e,* because *e* must be coprime to *ϕ(N)*. *e* must also be greater than 1 and less than *ϕ(N)*.

***Example: Determining* e**

ϕ(527) = (17 – 1)(31 – 1) = 16 × 30 = 480

let e = 77 ∵ 1 ≤ 77 ≤ 480 and gcd(77, 480) = 1

It is also helpful to use the Euclidean Algorithm, which determines the greatest common divisor (gcd) of two integers, a and b where $a > b$. The Euclidean Algorithm is done through a recursive process. First, solve the equation

$$a = q_1 b + r_1$$

for r_1 where $q_1 = \lfloor a / b \rfloor$. Then, one uses r_1 to solve the equation

$$b = q_2 r_1 + r_2$$

for r_2 where $q_2 = \lfloor b / r_1 \rfloor$. Then, one uses r_2 to solve the equation

$$r_1 = q_3 r_2 + r_3$$

for r_3 where $q_3 = \lfloor r_1 / r_2 \rfloor$. Continue solving for r_n using the pattern

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

until $r_n = 0$. When $r_n = 0$, the gcd is r_{n-1} . We can use this algorithm by letting $a = \phi(N)$ and $b = e$. This will help check for the gcd of large numbers. (Weisstein)

Example: Euclidean Algorithm

$\phi(N) = 480$ *and* $e = 77$

$480 = (\lfloor 480 / 77 \rfloor)(77) + r_1 \Rightarrow r_1 = 18$

$77 = (\lfloor 77 / 18 \rfloor)(18) + r_2 \Rightarrow r_2 = 5$

$18 = (\lfloor 18 / 5 \rfloor)(5) + r_3 \Rightarrow r_3 = 3$

$5 = (\lfloor 5 / 3 \rfloor)(3) + r_4 \Rightarrow r_4 = 2$

$3 = (\lfloor 3 / 2 \rfloor)(2) + r_5 \Rightarrow r_5 = 1$

$2 = (\lfloor 2 / 1 \rfloor)(1) + r_6 \Rightarrow r_6 = 0$

$r_6 = 0$ *and* $r_{6-1} = 1 \therefore gcd(77, 480) = 1$

Find *d*

The final step in the process is to find *d*. To find *d*, one just needs to find a value that satisfies the congruence relation

$$ed \equiv 1 \pmod{\phi(N)}$$

This can be solved by using the Extended Euclidean algorithm (assuming *e* fits the previous requirements). The Extended Euclidean algorithm is, as the name implies, and extension of the Euclidean algorithm. It begins with the Euclidean algorithm. Continue recursively with the algorithm until finding a value for *n* such that $r_n = 0$ and set value $a = n - 1$ for later use. Next, go back and solve for r_n for each equation. In other words, rearrange each equation to

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

Example: Rearrange Equations

$480 = (6)(77) + 18 \Rightarrow 18 = 480 + (-6)(77)$

$77 = (4)(18) + 5 \Rightarrow 5 = 77 + (-4)(18)$

$18 = (3)(5) + 3 \Rightarrow 3 = 18 + (-3)(5)$

$5 = (1)(3) + 2 \Rightarrow 2 = 5 + (-1)(3)$

$3 = (1)(2) + 1 \quad \Rightarrow 1 = 3 + (-1)(2)$

$a = n - 1 = 6 - 1 = 5$

After rearranging the equations, the algorithm uses them to find the modular inverse. To do so, it begins with

$$r_a = r_{a-2} + (-q_a)(r_{a-1})$$

Next, it replaces r_{a-1} with its equivalent, $r_{a-3} + (-q_{a-1})r_{a-2}$ as determined when the equations were rearranged, or . This makes the equation

$$r_a = r_{a-2} + (-q_a)(r_{a-3} + (-q_{a-1})r_{a-2})$$

or

$$r_a = (q_a q_{a-1} + 1)r_{a-2} + (-q_a)r_{a-3}$$

after distribution and combining like terms. It replaces r_{a-2}, r_{a-3}, … r_1 in the same manner. This causes the equation to end up being

$$r_a = ak_1 + bk_2$$

where k_1 is a positive integer and k_2 is a negative integer that is less than a. In terms of finding the modular inverse, this is very helpful.

When you have a value for e that is coprime to $\phi(N)$, that means that the gcd, and therefore r_a, is going to be 1; if we let $a = \phi(N)$ and $b = e$, this makes the equation

$$1 = \phi(N)k_1 + ek_2$$

If you take this equation modulus $\phi(N)$, it reveals the value of d. This is because

$$1 \equiv 1 \pmod{\phi(N)}$$

and

$$\phi(N)k_1 \equiv 0 \pmod{\phi(N)}$$

and

$$ek_2 \equiv e(\phi(N) + k_2) \pmod{\phi(N)}$$

(The above statement is true because $k_2 < 0$ and $|k_2| \leq \phi(N)$, therefore $k_2 \equiv \phi(N) + k_2 \pmod{\phi(N)}$. Therefore, the equation becomes

$$1 \equiv 0 + e(\phi(N) + k_2) \equiv e(\phi(N) + k_2) \pmod{\phi(N)}$$

The equation that we were trying to solve is

$$1 \equiv ed \pmod{\phi(N)}$$

and the equation that we have found is

$$1 \equiv e(\phi(N) + k_2) \pmod{\phi(N)}$$

Therefore, $d = \phi(N) + k_2$.

Example: Extended Euclidean Algorithm

$1 = 3 + (-1)(2)$

$1 = 3 + (-1)(5 + (-1)(3)) \Rightarrow 1 = (2)(3) + (-1)(5)$

$1 = (2)(18 + (-3)(5)) + (-1)(5) \Rightarrow 1 = (2)(18) + (-7)(5)$

$1 = (2)(18) + (-7)(77 + (-4)(18)) \Rightarrow 1 = (30)(18) + (-7)(77)$

$1 = (30)(480 + (-6)(77)) + (-7)(77) \Rightarrow 1 = (30)(480) + (-187)(77)$

$1 = (30)(480) + (-187)(77)$

$(30)(480) + (-187)(77) \equiv (480 + (-187))(77) \equiv 1 \pmod{480}$

$1 \equiv (293)(77) \pmod{480}$ *and* $e = 77 \therefore d = 293$

RSA Encryption and Decryption

Now we have found *N*, *e*, and *d*. These are separated into two different keys. The first key is the public key (*N*, *e*). The second key is the private key (*N*, *d*). If Bob is using these keys so that people can send a message to him, he publishes the public key, giving everyone access to it, while keeping the private key private.

Encryption

To encrypt a message, first one needs to convert the message to numbers through a predetermined method. Next, one breaks the message into blocks of a predetermined block size, often 214 to 342 bytes in the real world when message padding is included. For each block, use the following equation to encrypt it:

$$M^e \equiv C \pmod{N}$$

where M is the block, e is from the public key, C is the resulting cipher text, or encrypted text, and N is from the public key.

Exponentiation in modular arithmetic can be done traditionally. However, the numbers while become very large very quickly, especially in RSA encryption. There is another way that keeps numbers small that builds on the idea of multiplication in modular arithmetic. Consider the congruence relation $M^e \equiv C \pmod{N}$. To start finding M^e, one simply needs to find $M^1 \pmod{N}$, then $(M^1)^2 \pmod{N}$, then $(M^2)^2 \pmod{N}$, continuing this pattern of

$$M^n \equiv (M^{n/2} \pmod{N})^2$$

until $n = \lfloor log_2(e) \rfloor$. Keep track of each equation, as it may be used in the next step.

Next, split the exponent into powers of two. For example, if $e = 14$,

$$e = 2^1 + 2^2 + 2^3 = 2 + 4 + 8$$

This means that

$$M^{14} = M^2 \times M^4 \times M^8$$

We can then use the equations found above, multiplying each value found, then taking the result modulus N to find C.

Example: Encryption

let the public key (N, e) be (527, 77)

let the message M = 240

$77 = 2^0 + 2^2 + 2^3 + 2^6 = 1 + 4 + 8 + 64 \therefore 240^{77} = 240^1 \times 240^4 \times 240^8 \times 240^{64}$

$240^1 \equiv 240 \pmod{527}$

$240^2 \equiv 240 \times 240 \equiv 57600 \equiv 157 \pmod{527}$

$240^4 \equiv 157 \times 157 \equiv 24649 \equiv 407 \pmod{527}$

$240^8 \equiv 407 \times 407 \equiv 165649 \equiv 171 \pmod{527}$

$240^{16} \equiv 171 \times 171 \equiv 29241 \equiv 256 \pmod{527}$

$240^{32} \equiv 256 \times 256 \equiv 65536 \equiv 188 \pmod{527}$

$240^{64} \equiv 188 \times 188 \equiv 35344 \equiv 35 \pmod{527}$

$\therefore 240^{77} \equiv 240 \times 407 \times 171 \times 35 \equiv 584614800 \equiv 525 \pmod{527}$

$\therefore C \equiv 525 \pmod{527}$

Decryption

Decryption follows the same process. However, instead of using the public key (N, e) and the equation

$$M^e \equiv C \pmod{N}$$

we use the private key (N, d) and the equation

$$C^d \equiv M \pmod{N}$$

Example: Decryption

let (N, d) be (527, 293)

let the cipher text C = 525

$293 = 2^0 + 2^2 + 2^5 + 2^8 = 1 + 4 + 32 + 256 \therefore 525^{293} = 525^1 \times 525^4 \times 525^{32} \times 525^{256}$

$525^1 \equiv 525 \pmod{527}$

$525^2 \equiv 525 \times 525 \equiv 275625 \equiv 4 \pmod{527}$

$525^4 \equiv 4 \times 4 \equiv 16 \equiv 16 \pmod{527}$

$525^8 \equiv 16 \times 16 \equiv 256 \equiv 256 \pmod{527}$

$525^{16} \equiv 256 \times 256 \equiv 65536 \equiv 188 \pmod{527}$

$525^{32} \equiv 188 \times 188 \equiv 35344 \equiv 35 \pmod{527}$

$525^{64} \equiv 35 \times 35 \equiv 1225 \equiv 171 \pmod{527}$

$525^{128} \equiv 171 \times 171 \equiv 256 \equiv 256 \pmod{527}$

$525^{256} \equiv 256 \times 256 \equiv 65536 \equiv 188 \pmod{527}$

$\therefore 525^{293} \equiv 525 \times 16 \times 35 \times 188 \equiv 55272000 \equiv 240 \pmod{527}$

$\therefore M \equiv 240 \pmod{527}$

Conclusion

RSA encryption is a great example of how modular arithmetic is used in the real world. It shows that the math that we learn in school may actually be useful in our future careers. This exploration makes me want to make sure that I pay attention in math class, just in case what I learn is important later in life.

RSA encryption is also a very detailed and very thorough case study in modular arithmetic. It uses the most basic modular arithmetic, such as addition, multiplication, and exponentiation, but it also delves into theories and algorithms that involve modular arithmetic, such as Euler's Totient Theorem and the Extended Euclidean Algorithm. By trying to understand RSA encryption, I have learned a lot more about modular arithmetic than I originally thought I would.

This essay has been very beneficial to me for multiple reasons. The first is that is has taught me something, RSA encryption, that I hope to use in my future career. The second is that it has helped educate me about modular arithmetic and a few ideas and theories involved in modular arithmetic. Finally, this exploration has taught me that if I dedicate myself to figuring something out, I will eventually figure this out. This is because I originally struggled to understand Euler's Totient Theorem and was discouraged. In fact, I almost switched topics. However, after searching long and hard for a good explanation, I found one and was able to continue on with my topic.

This essay has also inspired some questions that I would like to investigate further. First, I would like to understand why the Euclidean algorithm works. I would also like to discover why Euler's totient can be used in the way it is used in this equation. I am also curious to look deeper into how RSA encryption is used in real life, such as in the military or in the government. Finally, I would like to look at other encryption methods and find out how they work.

In conclusion, I am glad that I had the opportunity to write this paper on RSA encryption because it gave me the opportunity to find out more about both RSA and modular arithmetic that I otherwise wouldn't have had because of lack of time.

Bibliography

Carlson, Andy. *About Enigma and Its Decryption.* California State University, San Bernardino, cse.csusb.edu/ykarant/cryptography/enigma/~andycarlson/enigma/about_enigma.html.

“Clock Position.” *Wikipedia*, upload.wikimedia.org/wikipedia/commons/thumb/d/d7/Horlo%C4%9Do_001.svg/1200px-Horlo%C4%9Do_001.svg.png.

Cruise, Brit. *Euler's Totient Function*. *Khan Academy*, Khan Academy, www.khanacademy.org/computing/computer-science/cryptography/modern-crypt/v/euler-s-totient-function-phi-function.

Der-Chyuan, Lou, et al. "Fast Exponentiation by Folding the Signed-Digit Exponent in Half." *International Journal of Computer Mathematics*, vol. 80, no. 10, Oct. 2003, pp. 1251-1259. EBSCO*host*, search.ebscohost.com/login.aspx?direct=true&db=syh&AN=11157830&site=ehost-live.

Earl, Richard. “Numbers and Codes.” Mathematical Institute. www.maths.ox.ac.uk/system/files/attachments/lecture3.pdf.

“The Euclidean Algorithm (Article).” *Khan Academy*, www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm.

GVSUmath, director. *The Euclidean Algorithm*. *YouTube*, YouTube, 12 Feb. 2014, https://www.youtube.com/watch?v=p5gn2hj51hs.

GVSUmath, director. *The Extended Euclidean Algorithm*. *YouTube*, YouTube, 12 Feb. 2014, www.youtube.com/watch?v=hB34-GSDT3k.

GVSUmath, director. *Multiplicative Inverses Mod n*. *YouTube*, YouTube, 12 Feb. 2014, https://www.youtube.com/watch?v=_bRVA5b4sb4.

Insall, Matt and Weisstein, Eric W. "Modular Arithmetic." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/ModularArithmetic.html

Matteo. “RSA Proof of Correctness.” *Cryptography Stack Exchange*, 13 June 2012, crypto.stackexchange.com/questions/2884/rsa-proof-of-correctness.

“Modular Arithmetic.” *Wikipedia*, Wikimedia Foundation, 12 Dec. 2017, en.wikipedia.org/wiki/Modular_arithmetic.

“RSA (Algorithm).” *Wikipedia*, Wikimedia Foundation, 17 Nov. 2017, simple.wikipedia.org/wiki/RSA_(algorithm).

Sinkov, Abraham, et al. *Elementary Cryptanalysis : A Mathematical Approach*. vol. 2nd ed. revised and updated by Todd Feil, Mathematical Association of America, 2009. Anneli Lax New Mathematical Library. EBSCO*host*, search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=452157&site=ehost-live.

Thomas Jankvist, Uffe. "A Teaching Module on the History of Public-Key Cryptography and RSA." *BSHM Bulletin: Journal of the British Society for the History of Mathematics*, vol. 23, no. 3, Nov. 2008, pp. 157-168. EBSCO*host*, doi:10.1080/17498430802304032.

United States, Congress, Barker, Elaine, et al. “Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography.” *Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography*, U.S. Department of Commerce, 2014.

Wagner. “RSA and the Chinese Remainder Theorem.” University of Berkley, California. people.eecs.berkeley.edu/~daw/teaching/cs70-f03/Notes/lecture12b.pdf.

Walter, Colin D. “Faster Modular Multiplication by Operand Scaling.” *SpringerLink*, Springer, Berlin, Heidelberg, 11 Aug. 1991, link.springer.com/chapter/10.1007/3-540-46766-1_26.

Weisstein, Eric W. “Euclidean Algorithm.” From *MathWorld*—A Wolfram Web Resource. http://mathworld.wolfram.com/EuclideanAlgorithm.html.

Weisstein, Eric W. "Totient Function." From *MathWorld*--A Wolfram Web Resource. http://mathworld.wolfram.com/TotientFunction.html