Number Theory Applied to RSA Encryption

New Mexico
Supercomputing Challenge

Final Report

April, 2015

Team #46
Los Alamos High School

Team Member:
Jovan Zhang

Teacher:
Duan Zhang

Mentors:
Duan Zhang
Adam Drew

Summary

RSA is the abbreviation of last names of three algorithm's inventors - Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1977. In RSA method, one creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. Anyone can use this public key to encrypt a message. However, when the public key is large enough, only the one who knows the prime numbers can feasibly decrypt the message. Currently, RSA method is regarded as the most reliable encryption method, and is widely used by large banking systems, credit card companies, and most internet service carriers, such as Google, Facebook, Yahoo, etc. In this project, I explored RSA method. First, I verified that RSA works for short messages. This provides a better understanding of the encryption method. To completely understand RSA encryption method, I studied the mathematical theory behind RSA method. To do so, I rigorously proved RSA encryption method using only elementary number theory and presented these proofs in the Appendix of this report. Based on the mathematical knowledge, I developed a Java program to demonstrate how RSA works, and overcame the limitation of message lengths. This Java code allows me to successfully encrypt and decrypt any messages. Furthermore, I tested the security level of RSA encryption by writing a Java program to perform a brute force attack on the encryption, which, essentially, is the only way to guarantee a crack on the RSA encrypted message. This program also tracks the time needed to crack RSA encryption by varying the length of the prime numbers. Through this calculation, I demonstrated that if large prime numbers are used to generate an encryption key, the modern RSA encryption can be extremely secure due to the massive amount of time which an attacker would need to crack the encryption. Finally, I presented a method to embed messages into pictures with my Java code. This steganography Java code manipulates the pixel brightness values in a image. Steganography is a useful way of sending encrypted information while avoids detection, which further enhances the security level of the encryption.

1 Introduction to RSA Encryption

Encryption is the process of intentionally making text illegible through a reversible process called encryption. The reversal process, called decryption, however requires a key only known to the intended readers, so that an eavesdropper, which may have malicious intent, cannot read the encrypted message.

In modern times, encryption and decryption have become more important. It is well known that every web link clicked or message sent over the internet can be read by virtually anyone if it is not protected. With private information, such as social security number, birthday, phone number, address, bank account all circulating around the web, an efficient and powerful form of encryption is necessary to protect citizens and governments from unintended accesses. Today, RSA encryption (hereinafter called RSA) is regarded as the most reliable method used to encrypt and decrypt sensitive information transmitted online, and widely used by large banking systems, credit card companies, and most internet service carriers, such as Google, Facebook, Yahoo, etc.

Conventional methods of encryption require the sender and receiver to meet in person to exchange keys and to agree on an encryption algorithm, which is a great limitation in today's world. Having to meet in person with another person thousands of miles away every time to establish a code is near impossible. RSA encryption provides a way to overcome this difficulty. To explain the idea of RSA, let us use a hypothetical example. Suppose Bob wants to send a gift to Alice through Eve, but he does not want Eve to see the gift. Bob puts the gift in a box and lock it. The conventional approach would require Bob to meet with Alice and give her a copy of the key. However, in RSA, Alice sends an open letter to Bob with instructions on how to make a lock to fit her key, but not how to make the key. In this way, if Eve intercepts and reads the message, she too, can send a locked box to Alice that only Alice can open, but Bob's package remains safe because Eve does not have the key to open it. Telling anyone in the world how to make a lock for someone's key is the concept of RSA.

To accomplish the encryption through a digitized world, RSA takes advantage of properties of numbers. RSA has two keys. One of them is public, and the other is kept secret. The public key is used to encrypt a message while the private key is used to decrypt the message. This allows anyone in the world to send an encrypted message, but only the intended receiver can decrypt the message once it has been encrypted.

In this project, my goal is to study the mathematical theory, application, and security of RSA algorithm by writing a Java program to explore it.

$0^9=$	0
$1^9=$	1
$2^9=$	512
$3^9=$	19683
$4^9=$	262144
$5^9=$	1953125
$6^9=$	10077696
$7^9=$	40353607
$8^9=$	134217728
$9^9=$	387420489

Figure 1: $a^9\,(\text{mod}10) = a\,(\text{mod}10)$.

1.1 A Simple RSA

Before we immerse ourselves into the rigorousness of number theory, let us first look at a very simple example of RSA. By a simple calculation, it is easy to prove that any number taken to the 9-th power retains its last digit, as shown in Fig. 1. This property can be exploited to form a simple RSA encryption and decryption.

Instead of taking a number to the 9-th power once, we can take it to the third power twice. The first time can be thought of as an encryption, and the second time is a decryption. For example, if our original message is 3, $3^3 = 27$. Taking the the last digit leaves us with the encrypted message, 7. To decrypt, we take 7 to the third power, $7^3 = 343$, which has the last digit 3, returning us to the original message. This is true for all numbers as shown in Fig. 2.

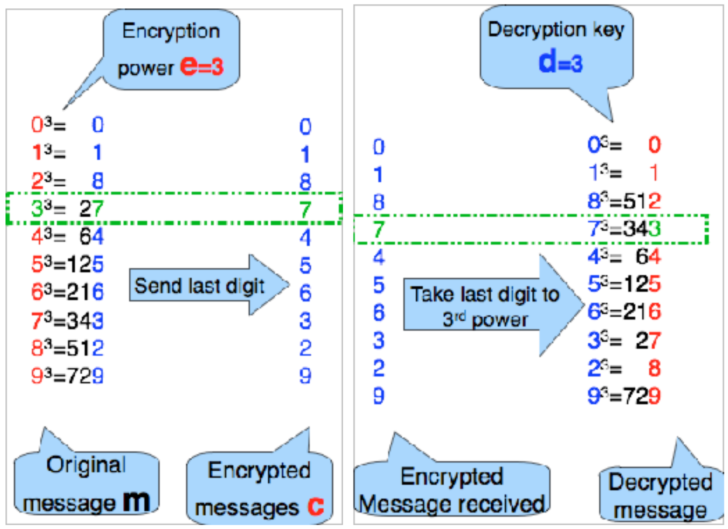

Figure 2: The encrypt and decrypt steps of a simple RSA algorithm.

2 Number Theory Behind RSA

RSA is rooted deeply in number theory. The example presented in the last section is a special case of a general RSA. As mentioned, RSA has two keys consisting of three numbers. We call them n, e, and d, where n is used in a modulo operation, (to divide a number and take the remainder), e is the encryption key, d is the decryption key. They are chosen as follows:

1. Choosing two distinct primes (called p and q.)
2. Multiplying p and q to get n.
3. Computing ϕ as: $(p-1) \times (q-1) = \phi$
4. Finding e such that $1 < e < \phi$, and e and ϕ are mutually prime.
5. Finding d such that $ed - 1$ is divisible by ϕ.

With these keys generated, n and e are broadcasted publicly, and d is kept secret for decryption.

In the simple example presented in the above, $p = 2, q = 5$, thus $n = 10, \phi = 4$. The only e and d that satisfy Steps 4 & 5 are $e = 3$ and $d = 3$. Although in this example e and d are the same, this is not generally true. We did not choose other example because this is the only one can be done before the calculation involves too large of integers for a pocket calculator. In a real applications, both e and n are usually a few hundred digit long; therefore the mathematical calculations required in these procedures are quite significant.

With this set of n, e and d, anyone can encrypt a numerical message m using the publicized n and e, by first taking the numerical message m to the e power to find m^e and then calculating the remainder of m^e divided by n. The remainder c is the encrypted message. Mathematically we express the procedure by

$$c = m^e \bmod n. \tag{1}$$

Such encrypted message c can then be sent over internet. The message can only be decrypted by using the decryption key d. To decrypt, one first takes c to d power to find c^d and then finds the remainder of c^d divided by n.

$$m_d = c^d \bmod n, \tag{2}$$

for $m < n$, such calculated m_d is the original message m, or $(m^e \bmod n)^d \bmod n = m$. This identity is the entire reason that RSA works [1]. The proof of this identity involves some nontrivial number theory steps and is presented in the Appendix for interested readers.

The requirement of $m < n$ costs a technical issue in the RSA application. The method of overcoming this issue is described in the next section.

3 RSA Implementation in A Java Code

3.1 Key Generation Algorithm

In my Java program the five steps described in the last section is followed. Two large numbers, not necessary primes, are picked. The two primes p and q are generated by using the *nextProbablePrime*() function in the BigInteger package of Java [2]. Although the *nextProbablePrime*() function in the package does not guarantee to return a prime number, the probability of error is less than 2^{-100}. With such obtained p and q, Steps 2 and 3 are performed.

To generate e, in my Java code e is chosen as the smallest prime number that is greater than $\phi/2$ by using *nextProbablePrime*() again. Since such chosen e is a prime, it satisfies Step 4. As Step 5, the decryption key d is generated by function *modInverse* in the BigInteger package. Occasionally, such generated $d = e$. In this case, we use *nextProbablePrime*() again to find another prime to be used as e, and Step 5 is then repeated to find d. The snapshot of my Java code output is shown in Fig. 3.

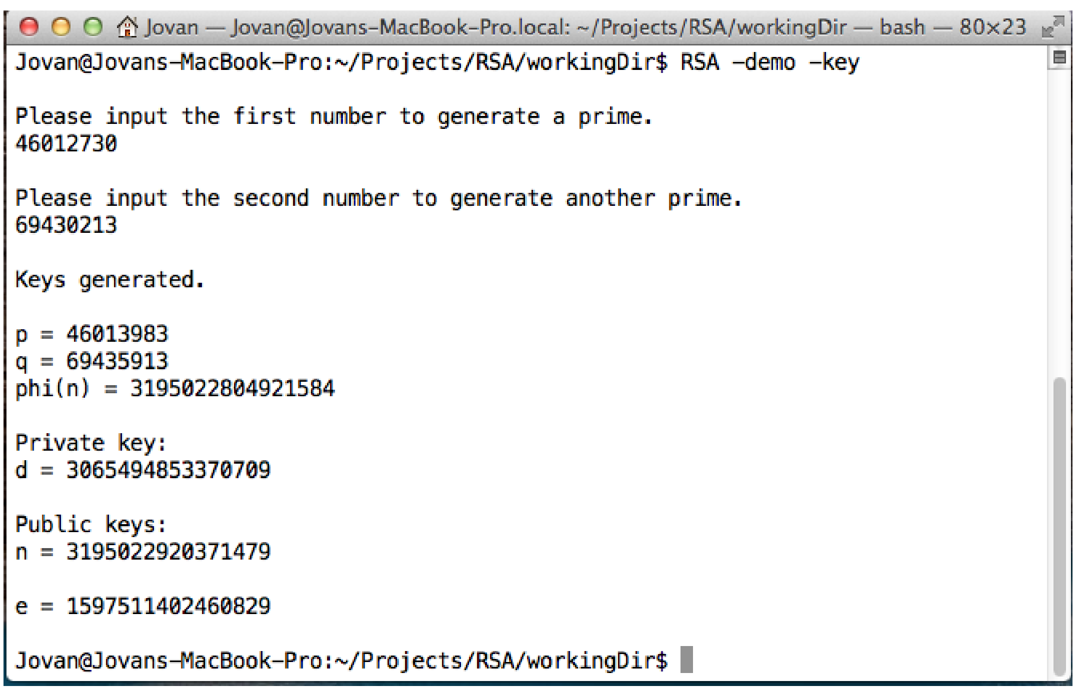

Figure 3: RSA key generation

3.2 Padding

RSA algorithm described above only deals with numbers, however our message are mostly written in words. To encrypt such a message, we need to translate text into a string of numbers. This process is called padding. ASCII (American Standard Code for Information Interchange) (shown in Fig. 4) allows text to be represented numerically. I wrote a Java method that converts each letter or symbol in a text into a integer using ASCII. For instance, the word “sun” is converted to a numerical string 115117110, with 115 corresponding to“s”, 117 to “u”, and 110 to“n”. However such padding encounters a problems. For instance the word “This” pads into “084104105115”. The computer automatically truncates this number into “84104105115” and I loose the first “0”. To avoid this problem we add 100 to a all numbers translated using ASCII.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Figure 4: The ASCII conversion table

3.3 Message Segmentation

For a typical sentence the padding process usually results in a long integer which could be much greater than n, and RSA algorithm cannot be applied directly. This issue is addressed in the segmentation process. The padded number is divided into segments that are less than n, and then encrypting and decrypting the segments separately. However this approach also encounters a problem. For example, the string 118341094217, if it is segmented into integers 118341 and 094217, Java would automatically cut the second integer into 94217, and we lose a digit, "0" that is important to the integrity of the message. To tackle this problem, my program tracks the length of each segment. Since the missing "0" always occurs at the beginning of a segmented integer, with the length saved, the missing "0"s can be added back during the recombining process after decryption.

3.4 Encryption and Decryption

As we have seen in Section 2, to encrypt, we take our message m to the power of e, as discussed in Section 3.1, and take the remainder when m^e is divided by n to obtain the encrypted message c. In my program, m was segmented, and each segment is encrypted separately. In my Java code, m and c are treated as arrays containing the segments of the messages. Similarly, to decrypt, we calculate the remainder when each element in array c^d is divided by n. We then reassemble the original numerical message with "0"s added if necessary, returning the padded message. Then we reverse the padding process using ASCII. Figure 5 illustrates a plaintext message, a padded message, and an encrypted message.

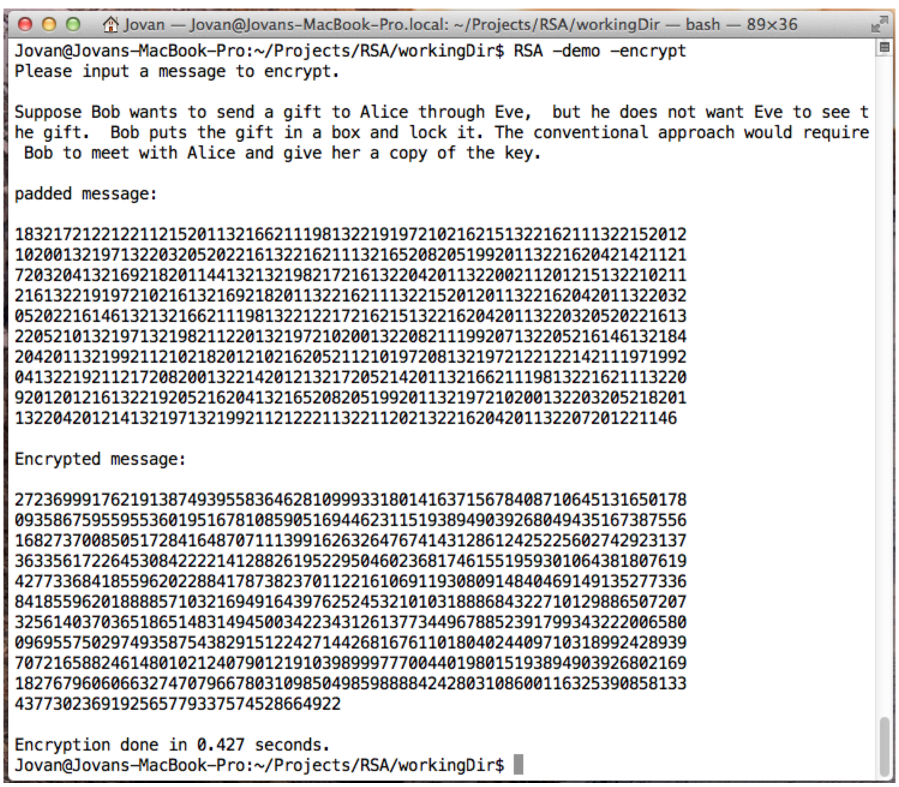

Figure 5: RSA encryption

The decryption process in shown in Fig. 6. The code outputs the decryption keys (d and n), the decrypted numerical string, and the decrypted and unpadded message.

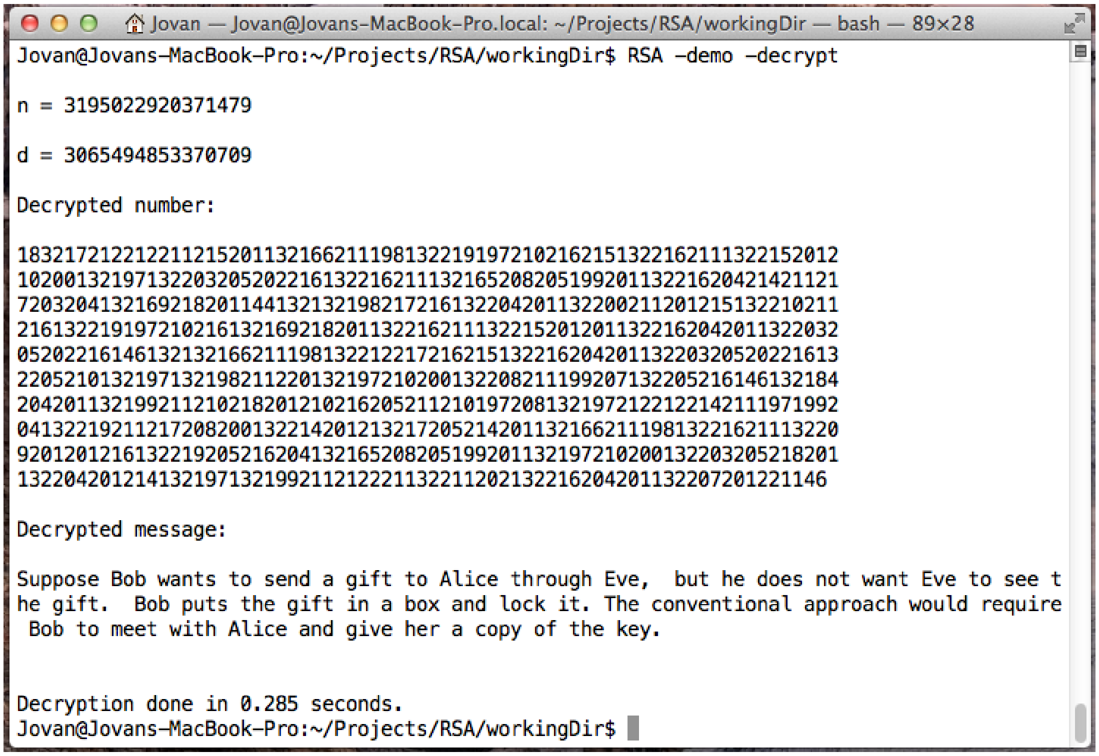

Figure 6: RSA decryption

4 Cracking RSA

RSA is popular because it is very easy to use and can be extremely secure. Its ease of use is quite obvious due to the public encryption as well as the private decryption keys. The security part, however, is more interesting. Since RSA broadcasts n and e publicly and $n = p \times q$, with p and q being two primes, knowing n there is a unique pair of primes p and q. If one finds p and q, with the publicized e, one can follow Steps 3 and 5 (skipping Step 4) in Section 2 to obtain the key d. In other words, the publicized n and e, in principle, contain everything that is needed to crack RSA. One ONLY needs to factorize n to crack an RSA encryption. However, this seemingly easy process is actually extremely difficult. The only current way to ensure the factorization is through a brute force attempt of each and every possible prime combination to find p and q from n. To demonstrate the difficulty, I wrote another Java program to attempt to crack RSA by this factorization method. For a given n, I instruct the computer to try every prime less than or equal to $\sqrt{n}$. I limit my search range to $\sqrt{n}$ to save the amount of calculation, because according to number theory, the smaller factor is always less than or equal to $\sqrt{n}$.

```
Jovan — Jovan@Jovans-MacBook-Pro.local: ~/Projects/RSA/workingDir — bash — 89×39
Jovan@Jovans-MacBook-Pro:~/Projects/RSA/workingDir$ RSA -demo -crack demoOut.png
Tried 0.3 million prime numbers in 17.996 seconds
Tried 0.6 million prime numbers in 37.058 seconds
Tried 0.9 million prime numbers in 56.549 seconds
Tried 1.2 million prime numbers in 1 minutes 16.198 seconds or 76.198 seconds
Tried 1.5 million prime numbers in 1 minutes 36.585 seconds or 96.585 seconds
Tried 1.8 million prime numbers in 1 minutes 56.566 seconds or 116.566 seconds
Tried 2.1 million prime numbers in 2 minutes 16.653 seconds or 136.653 seconds
Tried 2.4 million prime numbers in 2 minutes 37.775 seconds or 157.775 seconds
Tried 2.7 million prime numbers in 2 minutes 58.652 seconds or 178.652 seconds

Cracked! 3195022920371479 = 46013983 X 69435913

n = 3195022920371479

d = 3065494853370709

Decrypted number:

1832172122122112152011321662111981322191972102162151322162111322152012
1020013219713220320520221613221621113216520820519920113221620421421121
7203204132169218201144132132198217216132204201132200211201215132210211
2161322191972102161321692182011322162111322152012011322162042011322032
0520221614613213216621119813221221721621513221620420113220320520221613
2205210132197132198211220132197210200132208211199207132205216146132184
2042011321992112102182012102162052112101972081321972122122142111971992
0413221921121720820013221420121321720521420113216621119813221621113220
9201201216132219205216204132165208205199201132197210200132203205218201
132204201214132197132199211212221132211202132216204201132207201221146

Decrypted message:

Suppose Bob wants to send a gift to Alice through Eve,  but he does not want Eve to see t
he gift.  Bob puts the gift in a box and lock it. The conventional approach would require
 Bob to meet with Alice and give her a copy of the key.


Cracked in 3 minutes 4.228 seconds or 184.228 seconds.
Jovan@Jovans-MacBook-Pro:~/Projects/RSA/workingDir$ 
```

Figure 7: RSA Cracking

Figure 7 shows a cracking process. In this example the key d is 16 digit long. RSA encryption is cracked after trying about three million prime numbers. The cracking time is slightly more than three minutes.

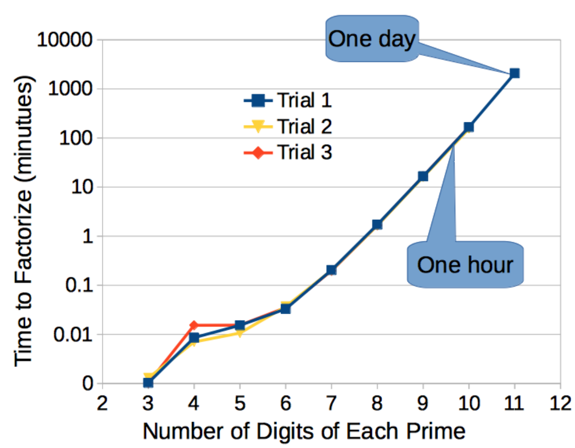

Figure 8: Length of p and q vs. cracking time.

Using my PC, the factorization time is plotted against the length of both p and q in Fig 8. For simplicity, p and q are the same length in these factorizations. In reality they do not have to be the same, but the smaller number will always be found first in the method I use, which starts with the smallest prime and advances. By timing the calculation, I found that my computer tries approximately 500,000 primes per minute.

For 10 digit long p and q, about 30,000,000 primes are tried.

Noting that the y scale is logarithmic, I predict that the cracking time grows exponentially as the length of p and q increase since the graph appears to be linear on the logarithmic scale for a key lengths above 6. The data in Fig. 8 can be exponentially fitted with the following correlation, as shown in Fig. 9,

$$\text{Time(hr)} = 7.41 \times 10^{-10} \times e^{2.2x}. \quad (3)$$

Using (3), I predict that the time needs to crack the n and e in my encryption program, shown in Fig. 10 is more than 10^{264} years. This is quite a long time comparing to 1.4×10^{10} years of the age of the universe. When these numbers are used to encrypt the message, the encrypted message is extremely secure. For the Yahoo website, it uses the 256-bit encryption. The key length is 78 digits long. If I use my program to crack its encrypted message, it will take more than 10^{28} years to crack with my PC.

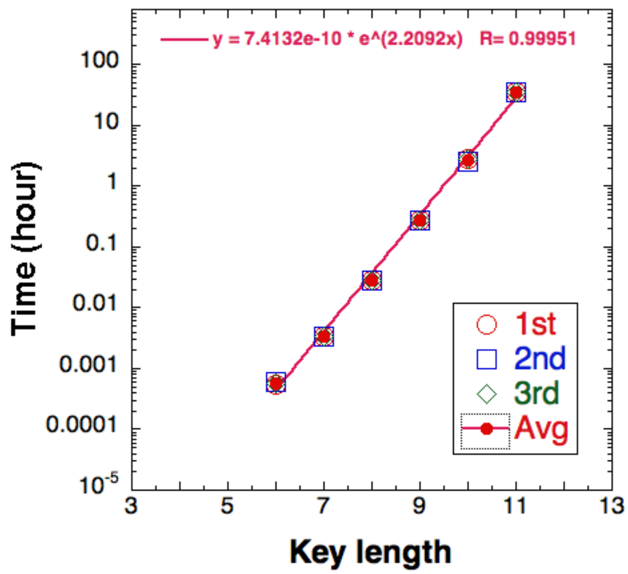

Figure 9: The correlation between cracking time and the key length

n:9310149675357111849678550519247138458421071411 69143
0761759153205501012873705880373217884066472300114808
927959424449230437322241695475919766580163668128 9192
0942615150970438391837087272926409869317606311942197
7884159158602339233050469523990351869860086954803003
8171215533058097220041623300521657452891308237702972
9018555397603412623334272261846604263270573072391026
1912549652709575890870985305270741538533876540113 4113
2396062750070270774678519540024659588795631691207844
2414931211917925061441418117059183994742807079569 9188
1847073163045282907764023464048332448450649629493

e:4655074837678555924839275259623569229210535705845 71
5380879576602750506436852940186608942033236150057404
4639797122246152186611208477379598832900818340644596
0471307575485219195918543636463204934658803155971098
8942079579301169616525234761995175934930043461006866
4468233602684174143626147905742124039991952483614885
5804661619667899245355527251646219146563428735928343
3170107729829313257367031113344059277937446968718874 2
7694929215707421258152644557232620948125830398471194
7832040974849216649405883937404709257838021885901758
304844341600109885917376102230116981623451773057 79

Figure 10: The “real” RSA public keys used in our program

5 Steganography

Steganography is the process of hiding information into a picture. Steganography and RSA work well together if one wants to avoid the detection of the transmission of an encrypted message. My steganography process starts duplicating a pixel in a picture four times to form another a picture with four times of its original size as shown in Fig. 11. With the help of photoMagic.java [3], I embed the encrypted message to the enlarged picture by adding message values, say 123456789, to RGB values of pixels 2, 3 and 4, while leaving the RGB values for pixel 1 unchanged as a reference. This process is shown in Fig. 11. A potential problem with this method, however, is if one of the RGBs value is too bright to begin with. For example, if we tried to add 5 to 253, we would get 258 which is outside the bounds of the pixel value. The solution is to ignore pixel brightnesses that are over 245. In other words, we tell the computer that if, say the red color value was over 245, the red color values of the corresponding four pixels contain no information, but the green and blue could still hold digits of the numerical string. To communicate the encrypted message, the enlarged pictures is sent. Since the message embedding process only changes the brightness values of the picture slightly, it is nearly indistinguishable to the human eye and therefore avoids detection. When the intended recipient gets the picture, the hidden encrypted message can be recovered by subtracting the RGB values of the reference pixel from the RGB values of the neighbor pixels. The RSA decryption process is then used to recover the transmitted message. Another advantage of this algorithm is that each pixel in the original picture can hold up to 9 digits of the encrypted string implying that a small picture can hold a lot of information.

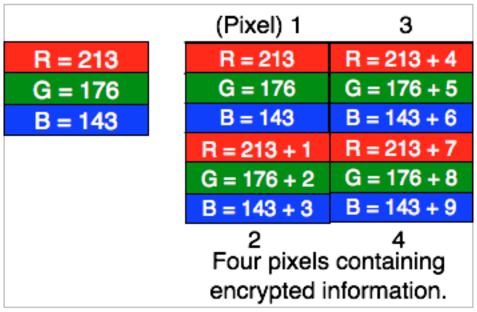

Figure 11: My steganographic algorithm.

The left of Fig. 12 shows a small picture used to embed a message. The entire Declaration of Independence is embedded into the right figure. In fact it only used only $< 5\%$ of the available pixels.

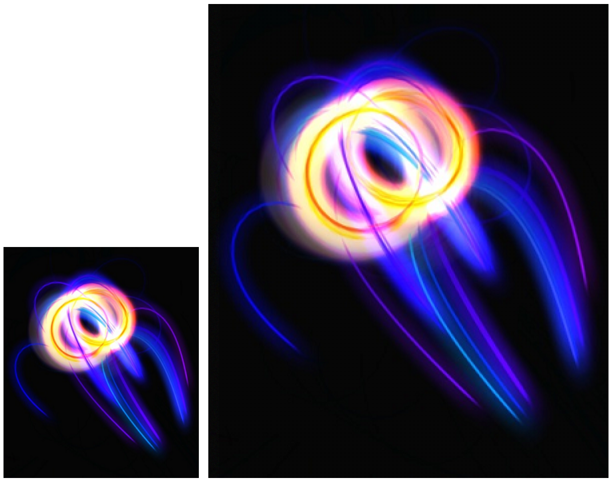

Figure 12: The larger picture contains an embedded message.

6 Conclusion

In this project, I have proved that RSA is a mathematically sound algorithm using elementary number theory. I wrote a Java program to demonstrate it, and dealt with various limitations of RSA and java such as message length and digit truncation. I have shown that RSA encryption is very secure. By attempting to crack it with a half million attempts per minute, it takes many many years to do so. Finally, if one wishes to hide information, encrypted or not, I have developed a steganographic process to efficiently embed numerical information into pictures without drastically changing the picture's appearance.

In the future, I may attempt an RSA encryption in binary where there will be more digits, but the picture brightness will only be varied by 1 or 0, even less distinguishable. Also, I would like to try more advanced ways to attack the RSA algorithm. For example I can look for possibilities in ϕ's. ϕ has the property of being between e and n, and also divisible by 4. If ϕ can be found, p and q can be calculated and hence RSA can be cracked.

Acknowledgments

I thank my mentor Duan Zhang for his frequent guidance on the project throughout this school year and providing many ideas on how to solve problems efficiently. I thank Horace Zhang for introducing me to photoMagic.java and Adam Drew for his guidance and constant lookout for new recourses. Finally, I would like to thank the Supercomputing staff for providing such a unique opportunity.

References

1. Long, Calvin T. (1972), "Elementary Introduction to Number Theory (2nd ed.)", Lexington: D. C. Heath and Company, LCCN 77171950

2. Niemeyer, Patrick. Knudsen Jonathan. "Learning Java". 101 Morris Street, Sebastpool, CA 95472: O'Reilly & Associates, Incorporated, May 2000.

3. "Index of / rs/cs126/LFSRassignment." Index of / rs/cs126/LFSRassignment. Princeton Univeristy, n.d. Web. 31 Mar. 2015.

4. Singh, Simon. "The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography." London: Fourth Estate, 1999. Print.

5. Boneh, Dan (1999). "Twenty Years of attacks on the RSA Cryptosystem". Notices of the American Mathematical Society 46 (2): 203-213.

6. Diffe, Whitfeild, and Martin E. Hellman. "New Directions in Cryptography." IEEE Transactions on Information Theory IT-22 (1976): 644-54. www.stanford.edu. Stanford University. Web. 22 Mar. 2015. <http://www-ee.stanford.edu/ hellman/publications/24.pdf>.

7. "List (Java Platform SE 7)." List (Java Platform SE 7). Oracle, n.d. Web. 22 Mar. 2015. <http://docs.oracle.com/javase/7/docs/api/java/util/List.html>.

8. Burnett, Steve, and Stephen Paine. RSA Security's Official Guide to Cryptography. New York: Osborne/McGraw-Hill, 2001. Print.

Appendix - Elementary Number Theory for RSA Algorithm

A.1 Lemmas

In this project, I will prove the functionality of RSA based on Fermat's Little Theorem. This proof however uses other lemmas and identifies that must be proven first to ensure a sound verification.

Notations:

- For integers a and b, notation $a \,|\, b$ means that b is divisible by a.
- For integers a and b, notation (a, b) denotes the greatest common divider of a and b.
- For integers a and b, $a \pmod{b}$ denotes the remainder of a divided by b.
- For integers a, b c, relation $a \equiv b \pmod{c}$ means $c \,|\, (a - b)$, which is equivalent to say that a and b have the same remainder upon division by c, or $a \pmod{c} = b \pmod{c}$. The notation "=" means two numbers are equal, and "≡" two numbers have the same remainder. For instance, we can say $2 = 14 \pmod{12}$, $2 \equiv 14 \pmod{12}$, and $26 \equiv 14 \pmod{12}$, but not $26 = 14 \pmod{12}$, because $14 \pmod{12}$ is 2 not 26.

Lemma 1: For integers a and b, if integer r is a remainder of a/b, i.e. $a = bq + r$ for some integer q, then $(a, b) = (b, r)$.
Proof: Since r is the remainder, there is an integer q such that $r = a - bq$, since (a, b) divides a and b, (a, b) divides both r and b, or (a, b) is a common divisor of b and r. Since (r, b) is the greatest common divisor of b and r, $(a, b) \leqslant (r, b)$.

On the other hand $a = bq + r$. That (r, b) divides r and b implies that (r, b) divides a and b. (r, b) is a common divisor of a and b. Since (a, b) is the greatest common divisor of a and b, $(r, b) \leqslant (a, b)$. Combining these two inequalities about (a, b) and (r, b), we have $(a, b) = (r, b)$. #

The Euclidean Algorithm: If $a > b$ are positive integers, the maximum common divider (a, b) can be found by the following algorithm.

$$\begin{aligned}
a &= bq_0 + r_0, && 0 \leqslant r_0 < b, \\
b &= r_0 q_1 + r_1, && 0 \leqslant r_1 < r_0, \\
r_0 &= r_1 q_2 + r_2, && 0 \leqslant r2 < r1, \\
r_1 &= r_2 q_3 + r_3, && 0 \leqslant r3 < r2, \\
&\cdots \\
r_{n-3} &= r_{n-2} q_{n-1} + r_{n-1}, && 0 \leqslant r_{n-1} < r_n \\
r_{n-2} &= r_{n-1} q_n + r_n, && 0 \leqslant r_n < r_{n-1}
\end{aligned} \tag{A.1}$$

Since $0 \leqslant r_n < r_{n-1}$, this procedure eventually ends with some integer m such that $r_m = 0$.
Proof: Since $r_m = 0$, $r_{m-1} \mid r_{m-2}$, $r_{m-1} = (r_{m-1}, r_{m-2})$. Using Lemma 1, $r_{m-1} = (r_{m-1}, r_{m-2}) = (r_{m-2}, r_{m-3}) = \cdots = (r_1, r_0) = (b, r_0) = (a, b)$. #

Bézout's identity: If $(a, b) = d$, then there are integers x and y such that $ax + by = d$.
Proof: Using the Euclidean Algorithm above, we have $d = r_{m-1} = r_{m-3} - r_{n-2} q_{n-1}$. Writing r_{m-3} in terms of r_{m-2} and r_{m-1}, and so on in the reverse order of the Euclidean Algorithm, we find the Bézout's identity.

Lemma 2 (the Euclid's Lemma): If $a \mid bc$ and $(a,b)=1$, then $a \mid c$.
Proof: Since $(a,b)=1$, there are integers x and y such that $ax+by=1$ (Bézout's identity). $c=xac+ybc$. Since $a \mid ac$ and $a \mid bc$, then $a \mid c$. #

Lemma 3 (The Cancellation Law): If $ax \equiv ay \pmod{p}$, and $(p,a)=1$, then $x \equiv y \pmod{p}$.
Proof: Given $ax \equiv ay \pmod{p}$, $p \mid a(x-y)$. Using Euclid's Lemma, $p \mid (x-y)$, that is $x \equiv y \pmod{p}$. #

Lemma 4 (The Multiplication Law): If $a \equiv b \pmod{p}$, and $x \equiv y \pmod{p}$, then $ax \equiv by \pmod{p}$.
Proof: We note $ax - by = (a-b)(x-y) + b(x-y) + (a-b)y$. Since $a \equiv b \pmod{p}$, and $x \equiv y \pmod{p}$ implies $p \mid (a-b)$ and $p \mid (x-y)$, p divides right hand side of the expression for $ax-by$, therefore $p \mid (ax-by)$, or $ax \equiv by \pmod{p}$. #

Corollary: For integers a, b and c, $(ab) \pmod{c} = \{a[b \pmod{c}]\} \pmod{c}$.
Proof: Let $r = b \pmod{c}$, then $r \equiv b \pmod{c}$. Since $a \equiv a \pmod{c}$, using the multiplication law, $ar \equiv ab \pmod{c}$, or

$$(ab) \pmod{c} = (ar) \pmod{c} = \{a[b \pmod{c}]\} \pmod{c}. \quad \#$$

Fermat's Little Theorem: If p is prime, and $p \nmid a$ then $a^{p-1} \equiv 1 \pmod{p}$, in other words, $a^{p-1} \pmod{p} = 1$.
Proof: Let us consider sequence

$$a, 2a, 3a, \cdots, (p-1)a \tag{A.2}$$

We first prove that if $1 \leqslant k_1 < k_2 \leqslant p-1$, then $k_1 a \pmod{p} \not\equiv k_2 a \pmod{p}$, or the remainder for divisor p is different for different elements in the sequence.

Suppose this is not true, then there are integers $k_1 \neq k_2$ such that $k_1 a \pmod{p} \equiv k_2 a \pmod{p}$, or $p \mid (k_2 - k_1)a$. Since $p \nmid a$, $p \mid (k_2 - k_1)$ according to the Euclid's Lemma, but $1 \leqslant k_1 < k_2 \leqslant p-1$, and $k_2 - k_1 < p-1$, a contradiction.

Let $r_k < q$ be the remainder of ka for divisor p, $r_k = ka \pmod{p}$. Since each r_k is different for different k, r_k takes values in between 1 and $p-1$ once and only once as k varies from 1 to $p-1$, and therefore the product of all remainders $n_1 \times n_2 \cdots n_{p-1} = (p-1)!$. Then using the multiplication law, we have

$$a \times 2a \times \cdots \times (p-1)a \pmod{p} \equiv n_1 \times n_2 \cdots n_{p-1} \pmod{p}. \tag{A.3}$$

or

$$a^{p-1}(p-1)! \pmod{p} \equiv (p-1)! \pmod{p}. \tag{A.4}$$

Since p is prime, $(p,(p-1)!) = 1$, using the cancellation law, we have $a^{p-1} \equiv 1 \pmod{p}$. #

Modulo Power Law: If $x \equiv n \pmod{y}$, then for a positive integer m, $x^m \equiv n^m \pmod{y}$, equivalently $x^m \pmod{y} = n^m \pmod{y}$,
Proof: If $x \equiv n \pmod{y}$, $x = iy + n$ for some integers i and y. Using binomial expansion we have

$$x^m = (iy)^m + m(iy)^{m-1}n + \cdots + C_m^k (iy)^{m-k} n^k + \cdots + m(iy)n^{m-1} + n^m,$$

therefore $y \mid (x^m - n^m)$, or $x^m \equiv n^m \pmod{y}$. #

Lemma 5 (A minor part of the **Chinese Remainder Theorem**): If $m \equiv x \pmod p$, $m \equiv x \pmod q$, and $(p, q) = 1$, then $m \equiv x \pmod{pq}$.
Proof: Given $m \equiv x \pmod p$, $m - x = ip$ for some integer i. Since $m \equiv x \pmod q$, $q \mid (m - x)$ or $q \mid (ip)$. Since $(p, q) = 1$, according to Euclid's lemma, $q \mid i$, or $i = kq$ for some integer k. $m-x = kqp$, or $m \equiv x \pmod{pq}$. #

A.2 Proof of RSA (The Algorithm)

Recall the following steps to generate the necessary keys for RSA:

1. Find two primes p and q.
2. Calculate $n = pq$ and $\phi(n) = (p - 1)(q - 1)$.
3. Find an integer e, such that $1 < e < \phi(n)$, and $(e, \phi(n)) = 1$.
4. Find an integer $d > 0$ such that $ed \equiv 1 \pmod{\phi(n)}$. This d is called modular inverse of e, denoted as $d = e^{-1} \pmod{\phi(n)}$

We note such modular inverse d exists because e is coprime to $\phi(n)$. Using Bézout's identity, there are integers x and y such that $xe + y\phi(n) = 1$. Since for any integer k we have $k\phi(n)e - ke\phi(n) = 0$, therefore $[x + k\phi(n)]e + (y - ke)\phi(n) = 1$, or $[x + k\phi(n)]e = 1 + (ke - y)\phi(n)$, $[x + k\phi(n)]e = 1 \pmod{\phi(n)}$. For a sufficiently large integer k, $x + k\phi(n) > 0$. The d can take this $x + k\phi(n)$. In this way we see solution for d is not unique, but the smallest positive d is less than $\phi(n)$.

To encrypt a number $m < n$: One calculates $c = m^e \pmod n$.

To decrypt: One calculates $c^d \pmod n$, since $c^d \pmod n = m$ as we now prove.

Proof: Since $c = m^e \pmod n$, $c \equiv m^e \pmod n$. Using the modulo power law we have $c^d \equiv m^{ed} \pmod n$. In other words c^d and m^{ed} have the same remainder $c^d \pmod n = m^{ed} \pmod n$.

Therefore to prove $c^d \pmod n = m$, it is sufficient to prove $m^{ed} \pmod n = m$, or $m^{ed} \equiv m \pmod n$. Since $ed \equiv 1 \pmod{\phi(n)}$, there is an integer h, such that $ed - 1 = h\phi(n) = h(p - 1)(q - 1)$.

$$m^{ed} = m^{ed-1}m = (m^{p-1})^{h(q-1)}m = (m^{q-1})^{h(p-1)}m. \tag{A.5}$$

If $p \nmid m$, using Fermat's little theorem, $m^{p-1} \equiv 1 \pmod p$. Using the modulo power law $(m^{p-1})^{h(p-1)} \equiv 1 \pmod p$. Because $m \equiv m \pmod p$, with the multiplication law, $(m^{p-1})^{h(p-1)}m \equiv m \pmod p$. Using relation (A.5), $m^{ed} \equiv m \pmod p$. If $p \mid m$, then $p \mid m^{ed}$, or $m^{ed} \equiv 0 \pmod p$ $\equiv m \pmod p$. In either case, we have $m^{ed} \equiv m \pmod p$.

Similarly, we can prove $m^{ed} \equiv m \pmod q$. Since p and q are prime, $(p, q) = 1$, using the Chinese remainder theorem, $m^{ed} \equiv m \pmod{pq}$. Since $n = pq$, $m^{ed} \equiv m \pmod n$. #

Appendix B - Java codes

Listing 1: Java Code

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.math.BigInteger;
import java.awt.Color;
public class RSA {
        BigInteger d;
        BigInteger e;
        BigInteger p;
        BigInteger q;
        BigInteger n;

//  options (encrypt, decrypt, weather to show pic or not, etc.)
        public static void main(String[] args) {
            long startTime;
        boolean showPic = false;
        boolean demo = false;
        boolean encrypt = false;
        boolean decrypt = false;
        boolean crack = false;
        boolean keyGen = false;
        int count = 0;
// show picture
        for(int i = 0; i <args.length; i++) {
                if(args[i].equals("-showpic")) {
                        showPic = true;
                        count++;
                }
        }
// demo- and option where the user has control over p, q, and the message
        for(int i = 0; i <args.length; i++) {
            if(args[i].equals("-demo")) {
                    demo = true;
                    count++;
            }
        }
// run the program encryption
        for(int i = 0; i <args.length; i++) {
                if(args[i].equals("-encrypt")) {
                        encrypt = true;
                        count++;
                }
        }
// run the program decryption
        for(int i = 0; i <args.length; i++) {
                if(args[i].equals("-decrypt")) {
                        decrypt = true;
                        count++;
                }
        }
```

```java
// attempt to crack the code, then show message
        for(int i = 0; i <args.length; i++) {
                if(args[i].equals("-crack")) {
                        crack = true;
                        count++;
                }
                }
// generate your own key
        for(int i = 0; i <args.length; i++) {
                if(args[i].equals("-key")) {
                        keyGen = true;
                        count++;
                }
                }
// program for key generation
                RSA myRSA = new RSA();
                if(keyGen) myRSA.setupRSA(demo);

//encrypt the message
                if(encrypt) {
                        String textFileName ="";
                        String inputPictureFileName = "demoIn.png";
                        String outputPictureFileName = "demoOut.png";
                        if(!demo) {
                                if(args.length < count+3) {
                                        System.out.println("Usage: ");
                                        System.out.println("RSA -encrypt [-showpic] inputTextFile, inputPngFile, outPngFile, or RSA -demo [-showpic] ");
                                        System.exit(0);
                                }
                                textFileName = args[count++];
                                inputPictureFileName= args[count++];
                                outputPictureFileName = args[count];
                        }
                        myRSA.encryptToPicture(textFileName, inputPictureFileName, outputPictureFileName, showPic, demo);

                }
                if(decrypt) { //decrypt
                        startTime = System.currentTimeMillis();
                        String inputPictureFileName;
                        String outputTextFileName = null;
                        if(demo && args.length==count) {
                                        inputPictureFileName ="demoOut.png";
                                        outputTextFileName = "demo.txt";
                        } else {
                                if(args.length < count+1) {
                                        System.out.println("Usage: ");
                                        System.out.println("RSA -decrypt inputPngFile, [outTextFile], or RSA -demo");
                                        System.exit(0);
                                }
                                inputPictureFileName = args[count++];
                                if(args.length > count) outputTextFileName = args[count];
                        }
```

```
				myRSA.decryptFromPicture(inputPictureFileName, outputTextFileName, demo);
				String line = myRSA.getElapsedTime(startTime);
				System.out.println("\n\nDecryption done in " + line + ".");
			}
			if (crack){	// crack
				startTime = System.currentTimeMillis();
				String inputPictureFileName;
				String outputTextFileName = null;
				if(demo && args.length==count) {
					inputPictureFileName ="demoOut.png";
					outputTextFileName = "demo.txt";
				} else {
					if(args.length < count+1) {
						System.out.println("Usage: ");
						System.out.println("RSA -crack inputPngFile, [outTextFile], or RSA -crack -demo ");
						System.exit(0);
					}
					inputPictureFileName = args[count++];
					if(args.length > count) outputTextFileName = args[count];
				}
				myRSA.crack();
				myRSA.decryptFromPicture(inputPictureFileName, outputTextFileName, demo);
				String line = myRSA.getElapsedTime(startTime);
				System.out.println("\n\nCracked in " + line + ".");
			}
		}


// pad
		public BigInteger pad(String s){
// it will pad String s1, which is your input message.
			String s1 ="";
			BigInteger paded;
			for (int i =0; i < s.length(); i++){
				char c= s.charAt(i);
// since the unpadding part requires all padded characters to be three digits long, we add 100 to all padded numbers to ensure that.
				int n = (int) c + 100;
				s1 = s1+Integer.toString(n);
			}
			paded = new BigInteger(s1);
			return paded;
		}

// the padded message must be shorter than n. So we break the message up into subStrings that are n-1 digits long
		public String[] breakup (BigInteger m){
			int digitOfn = n.toString().length();
			int numDigitInSub = digitOfn -1;
			String x = m.toString();
// string x counts the length of the message, so especially in picture encryption, the computer knows when to stop
			int digitsOfMessage = x.length();
			int NumSubstrings = digitsOfMessage/numDigitInSub+2;
```

```
                if(digitsOfMessage/numDigitInSub*numDigitInSub == digitsOfMessage) NumSubstrings -=1;
                String [] subStrings = new String [NumSubstrings];
                for(int i =0; i  < NumSubstrings; i++){
                        subStrings[i] = "";
                }
                for(int i = 0; i < digitsOfMessage; i++){
                                int isub = i/numDigitInSub;
                                subStrings[isub] += x.charAt(i);
                }
// the last string in array subStrings stores the length (in string) of the previous string.
                int lengthOfLastString = subStrings[NumSubstrings-2].length();
                subStrings[NumSubstrings-1]= Integer.toString(lengthOfLastString);
                return subStrings;
        }

// once the message is decrypted, it can be unpadded.
        public String unpad (BigInteger m){
                String s1 =m.toString();
                int length = s1.length();
// since each character is padded into 3 digit long numbers, this sends every three digits in, so it can be unpadded into the original message
                int numChar = length/3;
                String c3;
                int ascii;
                String unpadded = "";
                for (int i = 0; i < numChar; i++){
                        c3 = Character.toString(s1.charAt(3*i))
                           + Character.toString(s1.charAt(3*i+1))
                           + Character.toString(s1.charAt(3*i+2));
// after the three digits are read out, 100 is subtracted form that number to reverse that process used in padding
                         ascii = Integer.parseInt(c3)-100;
// the message is now unpadded
                         unpadded = unpadded+Character.toString((char) ascii);
                }
                return unpadded;
        }

// now that the subStrings are unpadded, we must reverse the breakup process
        public BigInteger combine(BigInteger[] message){
                String combined = "";
// since the sub string is a big integer, if a substring starts with zero, the program will naturally get rid of it.
                String zeros="";
                int digitOfn = n.toString().length();
                // this is used to see if numbers (zeros) are missing from the front of each substring.
                int numOfSubStrings = message.length;
                for (int i = 0; i < numOfSubStrings-2; i++){
                        zeros = "";
// the actualNumOfDigits command counts the number of digits there are supposed to be i each substring. This is used to see how many zeros are missing.
                        int actualNumOfDigits = message[i].toString().length();
                        int diffInDigit = digitOfn-1 - actualNumOfDigits;
                        for (int j = 0; j < diffInDigit; j++ ) zeros = zeros+"0";
                        combined =combined+zeros+message[i].toString();
                }
```

```java
                    int correctNumDigitOfLastSubString =  message[numOfSubStrings-1].
                        intValue();
                int atcualNumDigitOfLastSubString = message[numOfSubStrings-2].toString
                    ().length();
                int diffInDigit = correctNumDigitOfLastSubString -
                    atcualNumDigitOfLastSubString;
                zeros = "";
// once it has figure out how many zeros are missing in the front, subString[i] adds
    those zeros back.
            for (int i = 0; i < diffInDigit; i++ ) zeros = zeros+"0";
                combined = combined + zeros+message[numOfSubStrings-2].toString();
                    BigInteger combinedBigInt = new BigInteger(combined);
                    return combinedBigInt;
        }


/*
 * Encryption of picture
 */

        public void encryptToPicture(String textFileName,
                            String inputPictureFileName, String outputPictureFileName,
                                boolean showPic, boolean demo) {
                String s = "";
                String[] fromTxt;
                if(demo) {
                        System.out.println("Please input a message to encrypt.\n");
                        fromTxt = new String[1];
                        fromTxt[0] = readFromTerminal();

                } else {
                        fromTxt = readFormFile(textFileName);
                }
                long startTime = System.currentTimeMillis();
                String pubKey[] = readFormFile("public.dat");
                n =new BigInteger(pubKey[0]);
                e =new BigInteger(pubKey[1]);
                for(int i = 0; i < fromTxt.length-1; i++) s += fromTxt[i]+"\n";
                s += fromTxt[fromTxt.length-1];
                BigInteger m = pad(s);
                String[] subStrings =  breakup(m);
                BigInteger[] encrypted = encrypt(subStrings);
// for output only
                if(demo) {
                        System.out.println();
                        System.out.println("padded message:\n");
                        String ms = m.toString();
                        for(int i = 0; i < ms.length(); i++){
                                System.out.print(ms.charAt(i));
                                if((i+1)%70 ==0) System.out.print("\n");
                        }
                        System.out.print("\n");
                        System.out.print("\n");
                        String encryptedNum = "";
                        if(demo) System.out.println("Encrypted message: \n");
                        for(int i = 0; i < encrypted.length-1; i++){
                                encryptedNum += encrypted[i].toString();
                        }
                        for(int i = 0; i < encryptedNum.length(); i++){
```

```java
                                System.out.print(encryptedNum.charAt(i));
                                if((i+1)%70 ==0) System.out.print("\n");
                        }
                        System.out.println();
                        System.out.println();
                }
// for output only
                Picture pc =  embedMessageIntoPicture(inputPictureFileName, encrypted, showPic);
                if(showPic) pc.show();
                pc.save(outputPictureFileName);
                String line = getElapsedTime(startTime);
                System.out.println("Encryption done in " + line + ".");
        }

        public  Picture embedMessageIntoPicture(String inputPictureName, BigInteger[] message, boolean showPic) {
//turn BigInteger[] message into array of strings
                int arrayLength = message.length;
                String[] encryptedString = new String[arrayLength];
                String zeros ="";
                String s1 =Integer.toString(arrayLength);
//The first section of s1 contains info about total length of message array.
//The number of digit of length of the array must not exceed the digitOfn.
                int digitOfn = n.toString().length();
                if(digitOfn < s1.length()) {
                        System.out.println("Error: n is too small. Make it at least "+ arrayLength + " long.");
                        System.exit(0);
                }
                for (int j = 0;  j < digitOfn - s1.length(); j++) zeros = zeros+"0";
                s1 = zeros+s1;
                for (int i = 0; i < arrayLength; i++) {
                        zeros ="";
                        encryptedString[i] = message[i].toString();
                        int encryptedStringLength = encryptedString[i].length();
                        int numOfNeededZeros = digitOfn - encryptedStringLength;
                        for (int j = 0;  j < numOfNeededZeros; j++) zeros = zeros+"0";
                        encryptedString[i] = zeros +    encryptedString[i];
                        s1 += encryptedString[i];
                }
                Picture picInput = new Picture(inputPictureName);
                if(showPic) picInput.show();
                int width = picInput.width();
                int height = picInput.height();
                Picture picOutput = new Picture(2*width,2*height);
                int length = s1.length();
                int count = 0;
                int red1, red2, red3, green1, green2, green3, blue1, blue2, blue3;
                for(int i = 0; i < width; i++){
                        for(int j= 0;  j < height; j++){
                                Color picx = picInput.get(i,j);
                                int red = picx.getRed();
                                int green = picx.getGreen();
                                int blue = picx.getBlue();
                                red3 = red2 = red1 = red;
                                green3 = green2 = green1 = green;
                                blue3 = blue2 = blue1 = blue;
```

```java

                                        Color nc = new Color(red, green, blue);
                                // resizing the picture for encryption
                                picOutput.set(2*i,2*j,nc);
                        // Verifying the value is less that 245 so that the pixel's values will still be meaningful after embedding the message.
                                if(red < 245) {
                        // the message is now embedded into the corresponding three pixels from the original, similar for green and blue.
                                        red1 = red + Integer.parseInt(Character.toString(s1.charAt(Math.min(count++, length-1 ))));
                                        red2 = red + Integer.parseInt(Character.toString(s1.charAt(Math.min(count++, length-1 ))));
                                        red3 = red + Integer.parseInt(Character.toString(s1.charAt(Math.min(count++, length-1 ))));
                                }
                                if(green < 245) {
                                        green1 = green + Integer.parseInt(Character.toString(s1.charAt(Math.min(count++, length-1 ))));
                                        green2 = green + Integer.parseInt(Character.toString(s1.charAt(Math.min(count++, length-1 ))));
                                        green3 = green + Integer.parseInt(Character.toString(s1.charAt(Math.min(count++, length-1 ))));
                                }
                                if(blue < 245) {
                                        blue1 = blue +  Integer.parseInt(Character.toString(s1.charAt(Math.min(count++, length-1 ))));
                                        blue2 = blue +  Integer.parseInt(Character.toString(s1.charAt(Math.min(count++, length-1 ))));
                                        blue3 = blue +  Integer.parseInt(Character.toString(s1.charAt(Math.min(count++, length-1 ))));
                                }
                                Color nc1 = new Color(red1, green1, blue1);
                                Color nc2 = new Color(red2, green2, blue2);
                                Color nc3 = new Color(red3, green3, blue3);
                                picOutput.set(2*i+1,2*j,nc1);
                                picOutput.set(2*i,2*j+1,nc2);
                                picOutput.set(2*i+1,2*j+1,nc3);
                                }
                }
                return picOutput;
        }

/*
 * Decryption
 */

// private.dat is a file that contains the information needed to decrypt that is NOT published publicly

        public void decryptFromPicture(String inputPictureFileName, String outputTextFileName, boolean demo) {
                String s[] =  readFormFile("private.dat");
                n =new BigInteger(s[0]);
                d =new BigInteger(s[1]);
// the message is first extracted from the picture by subtracting the values of the three pixels from the original pixel value.
                Picture encryptedPic =  new Picture(inputPictureFileName);
// once the number are recovered, they will be decrypted, combined, and unpadded.
```

```
            BigInteger [] recoveredNum = extractMessageFromPicture(encryptedPic);
            BigInteger[] decrypted= decrypt(recoveredNum);
            BigInteger combined = combine(decrypted);
            String unpadded = unpad(combined);

            if(demo) {
                    System.out.println ("\nn = " + n );
                    System.out.println ("\nd = " + d);
                    System.out.println("\nDecrypted number:\n");
                    String ms = combined.toString();
                    for(int i = 0; i < ms.length(); i++){
                            System.out.print(ms.charAt(i));
// this will put 70 digits into a line, showing the decrypted number neatly
                            if((i+1)%70 ==0) System.out.print("\n");
                    }
                    System.out.print("\n");
            }

            System.out.println("\nDecrypted message: \n\n" + unpadded);
            if(outputTextFileName != null) writeToFile(outputTextFileName, false, unpadded );
    }

// this program tells the decrypt how to extract the message from picture
    public  BigInteger[] extractMessageFromPicture(Picture fromPicture) {
            int width = fromPicture.width()/2;
            int height = fromPicture.height()/2;
            int red1, red2, red3;
            int green1, green2, green3;
            int blue1, blue2, blue3;
            String firstString = "";
            int count =0;
            int digitOfn = n.toString().length();
            for(int i = 0; (i < width)&&(count < digitOfn); i++){
                    for(int j= 0;  (j < height)&&(count < digitOfn); j++){
                            Color picx0 = fromPicture.get(2*i,2*j);
                            Color picx1 = fromPicture.get(2*i+1,2*j);
                            Color picx2 = fromPicture.get(2*i,2*j+1);
                            Color picx3 = fromPicture.get(2*i+1,2*j+1);
                            int red0 = picx0.getRed();
                            red3=red2=red1=red0;
                            int green0 = picx0.getGreen();
                    green3=green2=green1=green0;
                            int blue0 = picx0.getBlue();
                            blue3=blue2=blue1=blue0;
                            if (red0 < 245 ){
                                    red1 = picx1.getRed();
                                    red2 = picx2.getRed();
                                    red3 = picx3.getRed();
                                    if(count < digitOfn) {firstString += Integer.toString(red1-red0);
                                            count++;
                                    }
                                    if(count < digitOfn) {firstString += Integer.toString(red2-red0);
                                    count++;
                                    }
```

```
                                        if(count < digitOfn) {firstString += Integer
                                            .toString(red3-red0);
                                        count++;
                                        }
                                }
                                if (green0 < 245 ){
                                        green1 = picx1.getGreen();
                                        green2 = picx2.getGreen();
                                        green3 = picx3.getGreen();
                                        if(count < digitOfn) {firstString += Integer
                                            .toString(green1-green0);
                                        count++;
                                }
                                if(count < digitOfn) {firstString += Integer.
                                    toString(green2-green0);
                                count++;
                                }
                                if(count < digitOfn) {firstString += Integer.
                                    toString(green3-green0);
                                count++;
                                }
                                }
                                if (blue0 < 245 ){
                                        blue1 = picx1.getBlue();
                                        blue2 = picx2.getBlue();
                                        blue3 = picx3.getBlue();
                                        if(count < digitOfn) {firstString += Integer
                                            .toString(blue1-blue0);
                                        count++;
                                }
                                if(count < digitOfn) {firstString += Integer.
                                    toString(blue2-blue0);
                                count++;
                                }
                                if(count < digitOfn) {firstString += Integer.
                                    toString(blue3-blue0);
                                count++;
                                }
                        }
                }
                }
                int numOfLines=Integer.parseInt(firstString);
                String[] extracted = new String[numOfLines];
                for(int i = 0; i<numOfLines; i++ ) extracted[i] ="";
                count = 0;
                int digit = 0;
                int line = 0;
                for(int i = 0; i < width; i++){
                        for(int j= 0;  j < height; j++){
                                Color picx0 = fromPicture.get(2*i,2*j);
                                Color picx1 = fromPicture.get(2*i + 1,2*j);
                                Color picx2 = fromPicture.get(2*i,2*j+1);
                                Color picx3 = fromPicture.get(2*i+1,2*j+1);
                                int red0 = picx0.getRed();
                                red3=red2=red1= red0;
                                int green0 = picx0.getGreen();
                        green3=green2=green1=green0;
                                int blue0 = picx0.getBlue();
                                blue3=blue2=blue1=blue0;
```

```
if (red0 < 245 ){
        count++;
        if(count > digitOfn)  {
                red1 =  picx1.getRed();
                line = digit/(digitOfn);
                if(line < numOfLines) extracted[line] +=Integer.toString(red1-red0);
                digit++;
        }
        red2 = picx2.getRed();
        count++;
        if(count > digitOfn)  {
                line = digit/(digitOfn);
                if(line < numOfLines) extracted[line] +=Integer.toString(red2-red0);
                digit++;
        }
        red3 = picx3.getRed();
        count++;
        if(count > digitOfn)  {
                line = digit/(digitOfn);
                if(line < numOfLines) extracted[line] +=Integer.toString(red3-red0);
                digit++;
        }
}

if (green0 < 245 ){
        green1 = picx1.getGreen();
        count++;
        if(count > digitOfn)  {
                line = digit/(digitOfn);
                if(line < numOfLines) extracted[line] +=Integer.toString(green1-green0);
                digit++;
        }
        green2 = picx2.getGreen();
        count++;
        if(count > digitOfn)  {
                line = digit/(digitOfn);
                if(line < numOfLines) extracted[line] +=Integer.toString(green2-green0);
                digit++;
        }
        green3 = picx3.getGreen();
        count++;
        if(count > digitOfn)  {
                line = digit/(digitOfn);
                if(line < numOfLines) extracted[line] +=Integer.toString(green3-green0);
                digit++;
        }
}
if (blue0 < 245 ){
        blue1 = picx1.getBlue();
        count++;
```

```
                                        if(count > digitOfn)  {
                                                line = digit/(digitOfn);
                                                if(line < numOfLines) extracted[line] +=Integer.toString(blue1-blue0);
                                                digit++;
                                        }
                                        blue2 = picx2.getBlue();
                                        count++;
                                        if(count > digitOfn)  {
                                                line = digit/(digitOfn);
                                                if(line < numOfLines) extracted[line] +=Integer.toString(blue2-blue0);
                                                digit++;
                                        }
                                        blue3=picx3.getBlue();
                                        count++;
                                        if(count > digitOfn)  {
                                                line = digit/(digitOfn);
                                                if(line < numOfLines) extracted[line] +=Integer.toString(blue3-blue0);
                                                digit++;
                                        }
                                }
                        }
                }
                BigInteger [ ] recoveredNum = new BigInteger[numOfLines];
                for(int i = 0; i < numOfLines; i++){
                        recoveredNum[i]=new BigInteger(extracted[i]);
                }
                return recoveredNum;
        }


// generate primes and calculate n, e, and, d.
    public void setupRSA(boolean demo) {
        if(demo) {
// during the demo, you can input your own numbers to create a prime.
                System.out.println("\nPlease input the first number to generate a prime.");
                String n1 =readFromTerminal();
                System.out.println("\nPlease input the second number to generate another prime.");
                String n2 = readFromTerminal();

// Add 1234 and 5678 to inputs and find next prime to ensure the numbers are large enough to ensure n is, or longer than 6 digits.
                p= new BigInteger(n1).add(new BigInteger("1234")).nextProbablePrime();
                q= new BigInteger(n2).add(new BigInteger("5678")).nextProbablePrime();
// If demo is not selected, the code will use the default inputs below.
        } else {
                p = new BigInteger("283938904804732743928794327984379327943279472309970147" +
                        "42913749832794832710732823839204890328409328904837483274732974893
```

```
                        " +
                    "
                        43249327498327897489327948279472391874903274093279041792307490327904709
                        " +
                    "
                        32479832789784973895748397548789578975474752482136306407584368
                        ");

            q= new BigInteger("
                3278927092347483276897489327893274890374093729074032704732017409231 8
                " +
                    "
                        7432870932789047382975983798576438765725486724726327187595743928574398
                        " +
                    "
                        873849356983472987957829485908098509438905492809584902795038099437507
                        " +
                    "
                        483587857934759437574357438509438504359043090946257569347098275432724 5
                        ");
        }
// calculation of n, e, and, d using p and q.
        p = p.nextProbablePrime();
        q = q.nextProbablePrime();
        if(q.equals(p)) q = p.nextProbablePrime();
        n = p.multiply(q);
        BigInteger one = new BigInteger("1");
        BigInteger two = new BigInteger("2");
        BigInteger pm1 = p.subtract(one);
        BigInteger qm1 = q.subtract(one);
        BigInteger phi = pm1.multiply(qm1);
        e = phi.divide(two).nextProbablePrime();
        if(e.compareTo(phi)==1) {
                System.out.println(" p and q are too small, try again.");
                System.exit(0);
        }
// do calculation to find d, phi(n), and e
        d = e.modInverse(phi);
        if(e.compareTo(d)==0) {
                e = e.nextProbablePrime();
                d = e.modInverse(phi);
        }
        String line = n.toString()+"\n" + d.toString();
        writeToFile("private.dat", false, line );

        line = n.toString()+"\n" + e.toString();
        writeToFile("public.dat", false, line );
        System.out.println ("");
        System.out.println("Keys generated.");
        if(demo) {
                System.out.println ("");
                System.out.println ("p = " + p);
                System.out.println ("q = " + q);
                System.out.println ("phi(n) = " + phi);
                System.out.println("\nPrivate key:");
                System.out.println ("d = " + d);

                System.out.println("\nPublic keys:");
                System.out.println ("n = " + n);
```

```
			System.out.println ("\ne = " + e+"\n");
		}


	}
// method of encrypting text
		public BigInteger[] encrypt(String[]  subStrings){
			BigInteger[] encrypted = new BigInteger[subStrings.length];
			for(int i = 0; i < subStrings.length; i++){
				BigInteger m = new BigInteger(subStrings[i]);

				encrypted[i] = m.modPow(e, n);
			}
			return encrypted;
		}
// method of decrypting text
		public BigInteger[] decrypt (BigInteger[] encrypted){
			int numOfSubStrings =encrypted.length;
			BigInteger [] decrypted = new BigInteger [numOfSubStrings];
			for(int i = 0; i < numOfSubStrings; i++ ){
				decrypted[i] = encrypted[i].modPow(d, n);
			}
			return decrypted;
		}


// when cracking, the only information of the key is public, so we can only use what is listed in the "public.dat" folder
		public void crack (){
			String s[] = readFormFile("public.dat");
			n =new BigInteger(s[0]);
			e =new BigInteger(s[1]);
			BigInteger factor1 = factorize (n);
			BigInteger factor2 = n.divide(factor1);
			BigInteger phi = factor1.subtract( new BigInteger ("1")).multiply(factor2.subtract( new BigInteger ("1")));
			System.out.println();
			System.out.println("Cracked! "+ n + " = " + factor1 + " X " + factor2);
			BigInteger d = e.modInverse(phi);
		String line = n.toString()+"\n" + d.toString();
// once the code has been cracked, the private data will be found, so we write this information to the "private.dat"  folder
		writeToFile("private.dat", false, line );
		}

		public BigInteger factorize(BigInteger input) {
//This method returns smallest factor of input.
// If input is a prime number, it returns 1.
			long start = System.currentTimeMillis();
			BigInteger zero = new BigInteger("0");
			BigInteger prime = new BigInteger ("1");
			BigInteger returnvalue = new BigInteger ("1");
			int count = 0;
			do {
				count++;
				prime = prime.nextProbablePrime();
				if (input.mod(prime).compareTo(zero) == 0){
					returnvalue = prime;
```

```
                                        break;
                                }
                                if(count%300000==0) {
                                        System.out.println("Tried " + count/1000000.0 + " million prime numbers in "+ getElapsedTime(start));
                                }
                } while (prime.multiply(prime).compareTo(input) <0);
                return returnvalue;
        }

        /*
         * handy tools.....................................................
         *
         */
// reads a file and runs it through the program to encrypt, decrypt, crack, etc.
                public String readFromTerminal() {
                BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
                String s ="";
                try {
                        s = bufferedReader.readLine();
                    } catch(IOException e) {
                        e.printStackTrace();
                        System.err.println("Error: " + e.getMessage());
                    }
                return s;
                }
                public String[] readFormFile(String filename) {
                         String [] read  = new String[0];
                         try{
// Open the file that is the first command line parameter
                                  FileInputStream fstream = new FileInputStream(filename);
// Get the object of DataInputStream
                                  DataInputStream in = new DataInputStream(fstream);
                                  BufferedReader br = new BufferedReader(new InputStreamReader(in));
// Read File Line By Line
                                  int lines = 0;
                                  while (br.readLine() != null)   {
                                          lines++;
                                  }
                                  in.close();
                                  fstream.close();
                                  FileInputStream fstream1 = new FileInputStream(filename);
                                  DataInputStream in1 = new DataInputStream(fstream1);
                                  BufferedReader br1 = new BufferedReader(new InputStreamReader(in1));
                                  read  = new String[lines];
                                  for(int i =0; i<lines; i++) {
                                          read [i] =br1.readLine();
                                  }
// Stop reading the file.
                                  in1.close();
                                  fstream1.close();
                                    }catch (Exception e){//Catch exception if any
```

```
                                        System.err.println("Error: " + e.getMessage());
                                }
                        return read;
                }

                public void writeToFile(String fileName, boolean append, String line) {
                        try {
                    PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(fileName, append)));
                    pw.println(line);
                    pw.close();
                    } catch (IOException e) {
                       System.out.println(e.getLocalizedMessage());
                    }
                }

// time how long it takes to perform the the selected operation
        public String getElapsedTime(long start) {
                String sfsec;
                long timeDiff = 0;
                long fsec = 0;
                long seconds=0;
                long minutes = 0;
                long hours = 0;
                timeDiff = System.currentTimeMillis() -start;
                seconds  = timeDiff/1000;
                fsec = timeDiff-seconds*1000;
                if(fsec<10) {
                        sfsec = "00"+fsec;
                } else if (fsec <100) {
                        sfsec = "0"+fsec;
                } else {
                        sfsec = ""+fsec;
                }
                if(seconds  >= 60.0) {
                        minutes = (int) seconds /60;
                        seconds = seconds  - minutes*60;
                        if(minutes >= 60) {
                                hours = (int) minutes/60;
                                minutes = minutes - hours*60;
                        }
                }
                String line = timeDiff/1000.0 + " seconds";
            if (minutes>0) line = minutes + " minutes " + seconds+"."+sfsec + " seconds or " + timeDiff/1000.0 + " seconds";
            if (hours>0) line = hours +" hours "+ minutes + " minutes " + seconds+"."+sfsec + " seconds or " + timeDiff/1000.0 + " seconds";
                return line;
        }

}
```