Proceedings of GREAT Day

Volume 2011 Article 3

2012

Are Your Credit Cards Safe from Me? Cracking RSA Cryptography

James Clark
SUNY Geneseo

Are Your Credit Cards Safe From Me? Cracking RSA Cryptography

James Clark

Introduction

We use cryptography every day. Governments use it to send secret communications, banks use it to send wire transfers, and we use it when we pay for anything with a credit card. If we want to send private information to anyone, cryptography is used in order to keep that information secret. So it is understandable that with such a high demand to send private information that is secure, it is important that the cryptosystem used cannot easily be cracked. In order to understand how we have reached this point, we shall look at the history of cryptography and how it has evolved to the systems we use today.

A Brief History and Evolution of Cryptography

The first cryptographic system widely used was the Caesar shift cipher. It was first documented as being used by Julius Caesar during the Gallic Wars to send military messages [10]. The idea behind this cipher is that the letters of the original message, or plaintext (represented by lower case letters), can be rewritten as different letters to create the encrypted message, or ciphertext (represented by upper case letters). An example of this is shown in Table 1. If we want to send the letter a, we would write D; to send b, we would write E; to send c, we write F; and so on and so forth. This example is the Caesar shift where the alphabet has been shifted by three places. Other examples of the Caesar shift can be seen in Table 2. The first one is an example where the ciphertext is completely random and the second one begins with the keyword or keyphrase Caesar Cipher (CAESRIPH) and then the rest of the ciphertext is alphabetical from the last letter of the keyphrase. Between these three types of Caesar shift algorithms, there are 26! permutations (or 403, 291, 461, 126, 605, 635, 584, 000, 000) of the Caesar cipher. Simon Singh notes in his book The Code Book that if we were able to check one key per second, it would take a billion times the lifetime of the universe to check every key in order to decipher the message [10]. Thus there became a need for cryptanalysts to break the cryptographer's codes and the war between the cryptographers and cryptanalysts began.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Table 1: Caesar cipher that is shifted three letters

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
G	Q	D	N	F	J	P	H	V	A	S	I	O	E	Z	B	C	X	L	W	R	Y	T	K	U	M
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
C	A	E	S	R	I	P	H	J	K	L	M	N	O	Q	T	U	V	W	X	Y	Z	B	D	F	G

Table 2: The first is an example of a random key while the second has the keyphrase *Caesar Cipher*.

The Caesar cipher was secure for centuries, until frequency analysis was created [10]. The idea behind frequency analysis is that certain letters are used more frequently than others and this fact can be used to crack the Caesar cipher. Let us assume that we are using the English language and have intercepted a message encrypted with the Caesar cipher. After examining a few texts in English, we are able to establish the frequency that each letter is used, such as in Figure 1. Since e is the most common letter (as seen by Figure 1), if the ciphertext's most common letter is Q then we can assume that Q=e. This pattern can be followed for the rest of the letters in the message until the key is found

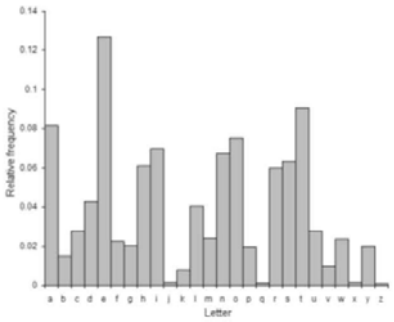
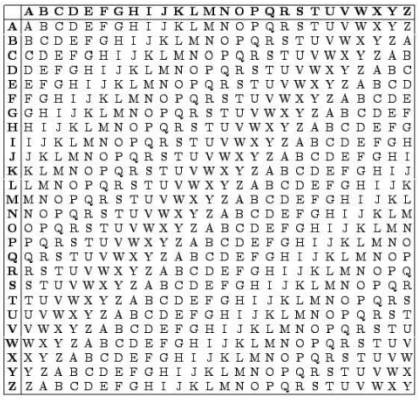


Figure 1: Frequency Analysis of the English Language

The next famous cipher was created in the early 1500s by a former diplomat Blaise de Vigenère[10]. His idea was to improve the Caesar cipher by encrypting the first letter with one Caesar cipher, the next letter with another Caesar cipher, the third letter with a third Caesar cipher, and so on and so forth. He created what is known as the Vigenère square (shown in Figure 2). A table and a keyword (such as *CODE*) would be agreed upon beforehand. To send a message, the first letter of the plaintext would be encrypted using the C row, the second letter with the O row, the third with the D row, and

the forth with the E row. Once the end of the keyword is reached the encryption would begin with the beginning of the code word again (in this case, with the letter C). The advantages of the Vigenère cipher is that with longer keywords it is more difficult to crack and it is also impervious to frequency analysis. Yet the cipher was eventually cracked in the 1850's by Edward Babbage because he was still able to find patterns [10].



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Figure 2: Virgenere Square

It was not until 1918 when the head of of cryptographic research for the U.S. Army, Major Mauborgne, created an uncrackable cipher using the Vigenère Cipher [10]. The idea behind the cipher is that a pad of random letters would be used as the keyphrase to encipher a message. Then once that message was sent the pad would be thrown away and a new pad would be used to encipher the next message. This cryptosystem, called the "one-time pad cipher", has three advantages. The first one is that frequency analysis cannot be used since it is a type of Virgenère cipher. The second advantage is the number of keys that would have to be tested is extremely large. For instance, if one message had 21 letters for its keyphrase, then there are 26^{21} or 518, 131, 871, 275, 444, 637, 960, 845, 131, 776 permutations of the one-time pad key, which is more than any human or computer could test in a lifetime. The third advantage is best shown by an example. Let us say that we have intercepted this message from our enemies: PEFOGJJRNUKCEIYVVUCXL. Assuming that it is a the length of the keyphrase is 21, we can either get the message attackthevalleyatdawn or defendthehillatsunsetor iwanttobeagreendragon. Now we can assume the person sending the message is not talking about how he or she wants to be a green dragon, but we do not know if our enemy is going to attack the valley at dawn or defend the hill at sunset. This shows us that we can get conflicting messages from the one-time pad cipher. In fact, we can get any message that is 21 letters long (assuming of course we are using a 21 letter keyphrase) which is what makes the one-time pad cipher uncrackable.

With the two world wars, there was a mechanization of cryptography with everything from decoder rings to the Enigma machine. All of these cryptosystems were eventually cracked except for the one-time pad cipher [10]. Yet the problem with the one-time pad cipher is that we have to send the key unencrypted, so we would either have to physically hand the key to the other person or send an unencrypted message that enemies could possibly intercept. Thus there was a need to be able to send encrypted messages without having to either communicate an encryption scheme with the other person ahead of time or send unencrypted keys that could potentially be intercepted.

Public Key Cryptography

As we saw with the one-time pad cipher, the problem is with the distribution of the key. If two people, let us say they are Alice and Bob, want to exchange messages, they must use a key which is a secret. So the problem becomes transmitting the secret key to the receiver in order to send the encrypted message safely. In other words, before the message is sent they must already have agreed upon a key.

Whitfield Diffie and Martin Hellman were working on the problem of public key cryptography and came up with a metaphor of what they wanted to accomplish. Let us say that Alice and Bob want to send a message to each other without Eve intercepting their message. Alice puts her message into a box, locks it with a padlock, and sends it to Bob. Bob cannot open it when he receives it because he does not have Alice's key, so he puts his own padlock on it and sends it back to Alice. Once Alice receives it, she unlocks her padlock leaving Bob's padlock to keep the box locked. She sends it back to Bob who can now unlock the box and read Alice's message. If Eve intercepts the box at any point that it is in transit between Alice and Bob, she will not be able to get into it [10].

There is a problem with this metaphor from a mathematical perspective that needs to be solved in order for the metaphor to work. The padlocks represents an encryption that is a one-way function,

which means that it is easy to compute forward and extremely difficult or impossible to go backward (or the inverse of the function is difficult/impossible to compute). So Alice first encrypts the message, then Bob encrypts the message, then Alice decrypts the message, then Bob decrypts the message so he can read it. The problem is that usually the last encryption needs to be the first one taken off, otherwise the message would not be readable [10]. The reason why the metaphor works though is because the padlocks are independent of each other. So they needed to find a one-way function that would work the same way the padlocks do on the box.

Diffie and Hellman solved the problem of key distribution in 1976 and outlined the solution in their paper "New Directions in Cryptography". This Diffie-Hellman encryption used the fact that

$$Y = \alpha^X \pmod q \ for\ 1 \le X \le q-1$$

is a one-way function for which it is extremely difficult to find its inverse. For instance, given any X we are able to calculate Y. Calculating X from Y where

$$X = \log_\alpha Y \pmod q \ for\ 1 \le Y \le q-1$$

is much more difficult due to the difficulty of computing logarithms mod q, which is why the technique is so secure [4].

So let us assume that Alice and Bob want to send a message to each other. Each of them must generate a random number from the set $\{1, 2, \ldots, q-1\}$ (X_A for Alice and X_B for Bob). Each of them are going to keep their respective numbers, X, a secret. Alice and Bob will communicate with each other and agree on an α and q. Next Alice will compute Equation 1 and Bob will compute Equation 2 and publish these results.

$$Y_A = \alpha^{X_A} \pmod q \quad (1)$$
$$Y_B = \alpha^{X_B} \pmod q \quad (2)$$

In order for Alice and Bob to communicate secretly, they calculate

$$K_{AB} = \alpha^{X_A X_B} \pmod q$$

and use that as their key.

Alice obtains K_{AB} by using Bob's published number, Y_B and calculates

$$\begin{aligned} K_{AB} &= (Y_B)^{X_A} \pmod q \\ &= \left(\alpha^{X_B}\right)^{X_A} \pmod q \\ &= \alpha^{X_B X_A} \pmod q = \alpha^{X_A X_B} \pmod q. \end{aligned}$$

Bob obtains K_{AB} in a similar way that Alice calculates K_{AB},

$$\begin{aligned} K_{AB} &= (Y_A)^{X_B} \pmod q \\ &= \left(\alpha^{X_A}\right)^{X_B} \pmod q \\ &= \alpha^{X_A X_B} \pmod q. \end{aligned}$$

If Eve was able to intercept all of the information sent between Alice and Bob (α, q, Y_A, and Y_B), in order to know K_{AB} she would have to calculate

$$K_{AB} = (Y_A)^{(\log_\alpha Y_B)} \pmod q.$$

Diffie and Hellman point out that if logarithms mod q could be easily computed then the cryptosystem would not work. While they do not have a proof of the this fact (or its converse), they could not find a way to compute K_{AB} from Y_A and Y_B unless they either have X_A or X_B. Thus they have found a new public key distribution where only one key needs to be exchanged and its use can be coupled with a directory of user information to authenticate Alice to Bob and Bob to Alice [4]. The only disadvantage is that we still had to communicate the key to the person that we want to send the messages.

It was not until 1977 that Ronald Rivest, Adi Shamir, and Leonard Adleman created a public key cryptography in which the receiver and sender did not have to communicate outside of the encrypted messages. We can understand their process with another metaphor where Alice wants to send Bob a message. Again, she is going to put the message in a box, but instead of putting her padlock on the box she is going to put on a combination lock where Bob knows the combination. So she goes to the store and buys the "Bob Combination lock," locks the box and sends the box to Bob. So when Bob receives the box, he can unlock it and read the message. Yet if Eve intercepts the box, she cannot get into it because she does not know the combination. So the lock itself would be the public key, the combination

would be the private key, the lock is the one-way function, and Alice and Bob never have to communicate outside the encryption. So how does RSA work?

Let us say that Alice wants to let other people, such as Bob, send her secure, encrypted messages that no one except Alice would be able to decrypt. She first selects two distinct primes, p and q, that are sufficiently large and random (e.g. not Mersenne primes) [3], where the two primes should not be close together, and both $p-1$ and $q-1$ have at least one large prime factor [6]. She then will multiply the primes together and call that number n (so $n = pq$). Alice then chooses a number e that is relatively prime to $\varphi(n)$ where $\varphi(n)$ is the Euler φ function and denotes the number of positive integers less than or equal to n that are also relatively prime to n (so $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1)$) [1]. $\varphi(n)$ and e are relatively prime if the greatest common divisor between $\varphi(n)$ and e is 1 (i.e. $g.c.d.(\varphi(n), e) = 1$) [1]. Once n is calculated and e is chosen, Alice has her public key and publishes these numbers in a directory. [3][6]

When Bob wants to send Alice a message, he wants to choose a ciphertext message that varies from user to user. So Bob wants to choose a plaintext to ciphertext algorithm that is uniform throughout the system. In order to do this he must first figure out what N-letter alphabet he wants to use (e.g. if he wants to use all of the letters, numbers, and symbols on the keyboard, then N would be 96.). He then chooses a $k \in \mathbf{N}$ such that $N^{k-1} < n < N^k$ (N^k should also be fairly large, Koblitz writes that it should be greater than 200 decimal digits [6]). Bob then takes the plaintext and splits it up into blocks of $k-1$ letters, or $(k-1)$-digit base-N integers. This means that Bob assigns numerical equivalents between 0 and N^{k-1} for each $k-1$ block. We similarly take ciphertext into units to be blocks of k letters in the N-alphabet. Therefore any plaintext message unit will be integers less than N^{k-1} and correspond to an element in $\mathbf{Z}/n\mathbf{Z}$ for any n. Since $n < N^k$, the image $f(P) \in \mathbf{Z}/n\mathbf{Z}$ can be uniquely written as a k-letter block. It is important to note that only the corresponding k-letter block integers that occur are integers less than n, so not all of the k-block integers will be used. We transform the $k-1$ block to a k block by taking each plaintext $k-1$ block, P, and calculate $C = P^e \pmod{n}$ where C is the ciphertext block. This number C can then be written as a k block ciphertext. [6]

Alice can decode the message by computing the e-th roots of the digits received (and they are unique since e is relatively prime to $\varphi(n)$) [3]. Alice can do this by calculating d, where $ed \equiv 1 \pmod{\varphi(n)}$, by using Euclid's algorithm on e and $\varphi(n)$ since raising to the d-th power is the same as taking the e-th roots. In fact, Alice can compute d as soon as e has been chosen and does not have to remember p and q. Consequently, if Eve intercepts the message, she can decrypt the message only if she can factor n into its two primes since she can repeat Alice's process to calculate d. Therefore the message is secure if n cannot be easily factored.[3][6]

This method does allow Alice and Bob to send messages without communicating the key to each other, which Diffie and Hellman could not offer with their encryption scheme. Yet what is stopping Eve from sending a message to Bob pretending to be Alice? To solve this problem Alice needs to know Bob's public key (n_B, e_B) and her private key (n_A, d_A). If $n_A < n_B$, Alice would send Equation 3 in order to send her signature M. Bob can verify that the message is from Alice by calculating Equation 4 where C is the ciphertext, d_B is part of Bob's decipher key, and e_A is part of Alice's encipher key. If $n_A > n_B$, Alice would then send Equation 5 and Bob will compute Equation 6. [6]

$$C = \left[M^{d_A} \pmod{n_A}\right]^{e_B} \pmod{n_B} \quad (3)$$

$$M = \left[C^{d_B} \pmod{n_B}\right]^{e_A} \pmod{n_A} \quad (4)$$

$$C = \left[M^{e_B} \pmod{n_B}\right]^{d_A} \pmod{n_A} \quad (5)$$

$$M = \left[C^{e_A} \pmod{n_A}\right]^{d_B} \pmod{n_B} \quad (6)$$

Therefore the only way that Eve could read Alice's messages or even send messages as Alice is if she can factor n_A. So how hard can it be to factor n_A into its two prime factors?

Methods of Factorization to Crack RSA Cryptography

If we want to crack RSA cryptography we must be able to figure out the prime factorization of n. It is important to note that there are ways to manipulate the particular execution of RSA in certain cases that would allow Eve to learn the secret plaintext intended for Alice by using a chosen-ciphertext attack [7], but this is outside the scope of the project and we will be concentrating on methods of factorization to crack RSA cryptography. Thus, the most intuitive way to approach the problem is

to create a list of primes less than $\sqrt{n}$ (e.g. Sieve of Eratosthenes or using primality testing on each number less than $\sqrt{n}$) and start dividing n by these primes to see if we get another prime q. Yet if n is sufficiently large it may take an extremely long time to arrive at n's factorization. As a result, we will be looking at different factorization methods in order to find the prime factors of n.

Pollard's Rho Method of Factorization

The first method of factorization that we will be looking at is Pollard's rho method of factorization, which is the simplest factorization algorithm and substantially faster than the trial division of primes less than $\sqrt{n}$ [6]. First, we choose an easily evaluated map $f : \mathbf{Z}/n\mathbf{Z} \rightarrow \mathbf{Z}/n\mathbf{Z}$, which means we choose a fairly simple polynomial mod n with integer coefficients. It is best that this function maps to itself in a disjointed manner (and should therefore not be linear and 1-to-1)[6], some examples of a function that we would use are $f(x) = x^2 + 1 \ (mod\ n)$ or $f(x) = 2x^2 + x + 5 \ (mod\ n)$. Next, we choose a particular starting values $x = x_0$, where x_0 can either be selected or randomly generated, and compute consecutive iterations of f (meaning that $x_{i+1} = f(x_i)$, for $i = 0, 1, 2, \cdots$). Finally, we compare the different x_i's in order to find two with different residue classes modulo n that are the same residue class modulo a divisor of n. When such an x_i and x_j are found, we compute the $g.c.d.(x_j - x_i, n)$ to find a prime factor n (meaning not 1 or n) and can then find the second prime factor by dividing n by the first prime factor. Since we are finding a point in the sequence where $x_i \equiv x_j$ we see that this pattern, which can be seen in Figure 3, resembles a Greek letter ρ and is where the method obtained its name. [6][8]

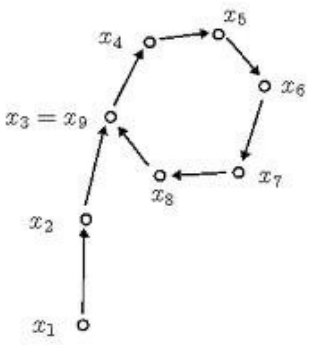


Figure 3: Where the Rho Method gets its name

Yet when j becomes substantially large, it is time consuming to compute $g.c.d.(x_j - x_i, n)$ for each $i < j$. We can minimize our work by computing one $g.c.d.$ for each j. When we have found a i_0 and a j_0 such that $x_{i0} \equiv x_{j0} \ (mod\ r)$ for some r that is a divisor of n (i.e. $r|n$), we have the relation $x_i \equiv x_j \ (mod\ r)$ for any pair of indices i, j such that $j - i = j_0 - i_0$. So when $x_i \equiv x_j \ (mod\ r)$, then $x_{i+1} = f(x_i) \equiv f(x_j) = x_{j+1} \ (mod\ r)$ and the sequence x_i becomes periodic $mod\ r$ with period $j - i$. One method then to reduce the number of $g.c.d.$s that have to be calculated is after we compute x_j we suppose that j is an $(h + 1)$-bit integer, which means that $2^h \leq j < 2^{h+1}$. Let i be the largest h-bit integer, which means $i = 2^h - 1$. We compute $g.c.d.(x_j - x_i, n)$ for these particular i, j. If the $g.c.d.$ gives a nontrivial divisor of n then we have our answer, if not we make the same calculations for $j + 1$. Another method of calculating $g.c.d.$s is to let b equal the period of the sequence (i.e. $b = j - i$), then $x_k \equiv x_l \ (mod\ r)$ whenever $k \equiv l \ (mod\ b)$, $k \geq i$, and $l \geq i$. Therefore, if we let k be the least multiple of b where $k \geq i$ and let $l = 2k$, then $x_k \equiv x_l \ (mod\ r)$. As a result, we can calculate $g.c.d.(x_{2i} - x_i, n)$ to find a nontrivial factor of n. [6][8]

The advantage of these two methods of calculating the $g.c.d.$ is that we need to only compute one for each j, but the drawback to this is that it probably will not detect the first time there is a j_0 where $g.c.d.(x_{j0} - x_{i0}, n)$ is a nontrivial factor of n for some $i_0 < j_0$. Nevertheless, we will find a x_j and x_k whose difference has a common factor with n in less time. Also, the method may be expected to disclose the smallest prime factor p of n in roughly $\sqrt{p}$ cycles, and is therefore faster than trial division for large n. It is important to note that the divisor obtained by this method may either be a prime factor, a composite number, or even n itself. In this situation when it is not a prime factor we would have to start over with a new x_0, a new function f, a new way to compute the $g.c.d.$, or any combination of these three inputs. [6][8]

Fermat's Method of Factorization, Part I

The second method of factorization that we will look at relies on the theorem that every odd number in $\mathbf{Z}$ can be represented by the difference of two squares ($\forall ac\mathbf{Z}$, $(a+1)^2 - a^2 = a^2 + 2a + 1 - a^2 = 2a + 1$). Thus we can represent the n from the public key in RSA cryptography as

$$n = pq = b^2 - c^2 = (b+c)(b-c).$$

Given this fact, we can let $p = (b+c)$ and $q = (b-c)$ [so $b = \frac{(p+q)}{2}$ and $c = \frac{(p-q)}{2}$].

$$b = \lfloor \sqrt{(n)} \rfloor + 1, \quad (7)$$

(where $\lfloor\ \rfloor$ is the floor function) and evaluate

$$b^2 - n = c^2. \quad (8)$$

If c^2 a perfect square, then we have found b and c and have therefore found p and q (since p and q defined in terms of b and c). If c^2 is not a perfect square, we evaluate b = $[\sqrt{(n)}]$ + 2 and compute Equation 8 again to see if the new c^2 is a perfect square. We continue this iterated process until c^2 is a perfect square and therefore found our factors. [6]

If p and q are close together, then $c = \frac{p-q}{x}$ is small and b is slightly larger than $\sqrt{n}$. Therefore we will be able to find the factorization of the n after a few iterations of this process. So what happens if p and q are not close together? The Fermat method will eventually find p and q, but only after trying a large number of iterations of b, but there is a way to minimize the number of calculations and find our factorization. First, let $k\epsilon$N be small and

$$b = \lfloor \sqrt{kn} \rfloor + 1. \quad (9)$$

We then evaluate

$$b^2 - kn = c^2 \quad (10)$$

and see if c^2 is a perfect square. We used the same process as before to find a b where Equation 10 is a perfect square. Once such a b is found, we know that $(b + c) = (b - c) = kn$ and therefore know that $b + c$ and n have a nontrivial common factor. We can then find this factor by computing $g.c.d.(b + c, n)$ to get one of our prime factors. This generalized Fermat method works quicker than the method outlined with Equations 7 and 8 when q is close to kp, and therefore reduces the number of b's that need to be tried to find the factorization of n. This method can then be generalized even further and leads to a more efficient factoring method. [6]

Fermat's Method of Factorization, Part II

An element $z\epsilon\ /n$ is a quadratic residue modulo n if there exists an $x\epsilon\ /p$ such that $x^2 = z\ (mod\ n)$, and means that x is a square root of z. Also, if $n = pq$ were p and q are distinct primes, then every quadratic residue modulo n has exactly four square roots. Therefore, if we are given any $b^2 \equiv c^2\ (mod\ n)$ where $b \not\equiv c\ (mod\ n)$, we can compute a non-trivial factor since Equation 11 implies Equation 12.

$$b^2 \equiv c^2 \pmod{n} \quad (11)$$

$$b^2 - c^2 \equiv 0 \pmod{n}$$

$$(b + c)(b - c) \equiv 0 \quad (12)$$

We can then fin d a factor of n by computing $g.c.d.(b + c, n)$ or $g.c.d.(b - c, n)$ since $n | b^2 - c^2$ and $n \nmid (b + c)$ or $n \nmid (b - c)$ because $b \not\equiv \pm c\ (mod\ n)$. Therefore, $g.c.d.(b + c, n) = p$ is a proper factor of n and $q\frac{n}{p}$ is divisible by $g.c.d.(b - c, n)$. [5][6]

Thus, this quadratic sieve algorithm needs to find values for b and c such that their squares are equal modulo n. Yet if n is sufficiently large, a random selection of b where the least positive residue of $b^2\ (mod\ n)$ is a perfect square is improbable and is necessary to generalize the method to allow flexibility in our choice of b's to evaluate. Thus we want to choose several b's with the condition $b_i^2\ (mod\ n)$ is a product of small prime powers so that a subset of them will give a b whose square is congruent to a perfect square modulo n when multiplied together. [5][6]

So let B be a factor base, which is set $B = \{p_1, p_2, \dots, p_k\}$ of distinct primes (except p_1 may be -1). We then must find integers b such that, for any given n that we are factoring, $b^2\ (mod\ n)$ is the least absolute residue and can be written as a product of numbers from B (found by trial division and primarily testing). We will call these numbers B-numbers. Also, b_i should be greater than $\sqrt{n}$ to ensure that$b_i^2\ (mod\ n) = \prod_{j=1}^{h} a_{ij}$ and the jth component of $\beta_i = \alpha_{ij}\ (mod\ 2)$, so

$$\beta_j = \begin{cases} 0\ if\ \alpha_{ij}\ is\ even \\ 1\ if\ \alpha_{ij}\ is\ \ odd \end{cases}$$

So consider when we have some set of B-numbers b_i such that the corresponding vectors $\underset{\beta_i}{\rightarrow}$ = {β_{i1}, . . . , β_{ih}}add up to the zero vector in F_2^h. This means that the product of least absolute residues of b_1^2equals the product of even powers of all of the p_j in *B*. So if for each *i* we let a_i be the least absolute residue of b_1^2 (*mod n*) and write $a_i = \prod_{j=1}^{h}(p_j)\, aij$ *we get*

$$\prod a_i = \prod_{j=1}^{h}(p_j)^{\sum_i \alpha_{ij}} \quad (13)$$

$$= \prod_{j=1}^{h}\left[(p_j)^{\frac{1}{2}\sum_i \alpha_{ij}}\right]^2 \quad (14)$$

Since the exponent of each p_j in Equation 13 is an even number, we can rewrite the product as a square and get Equation 14. Hence, when we set $b = \prod_i b_i \ (mod\ n)$ and $c = \prod_j \left[(p_j)^{\frac{1}{2}\sum_i \alpha_{ij}}\right] \ (mod\ n)$, where both *b* and *c* are least positive residues, we get two numbers whose squares are congruent modulo *n*. [5][6]

Of course, this fails if b ≡ ±c (*mod n*) and we must start over again with another collection of B-numbers whose corresponding vectors sum to zero (this will happen if we choose B-numbers less than $\sqrt{\frac{n}{2}}$ since all of the vectors are zero-vectors and we get a trivial congruence). Yet if we choose b_i randomly we can expect that $b \equiv c$ (*mod n*) (up to ±1) at most 50% of the time. The reason for this is because any square modulo *n* has 2^r square roots if *n* has *r* distinct prime factors. Therefore a random square root of b^2 has $\frac{2}{2^r}$ chance of being ±*b* (which is less than 50% since r ≥ 2). So when we have $b^2 \equiv c^2$ (*mod n*) where $b \not\equiv \pm c$ (*mod n*) we can find a nontrivial factor of n by computing *g.c.d.*(*b* + *c*, *n*). Koblitz points out that if we go through this procedure of finding *b* and *c* until we find a pair that gives us a nontrivial factor of n, there is at most a 2^{-k} probability that it will take more than *k* tries. [6]

We can choose our factor base *B* and our *B*-numbers b_i with a few different methods. One method is to start with *B* consisting of the first *h* − 1 primes, let $p_1 = -1$, and choose random b_i's until several are found that are *B*-numbers. Another method is to start choosing some b_i where b_i^2 (*mod n*) is a least absolute residue and is small in absolute value (one way to do this is to choose b_i close to $\sqrt{kn}$ where *k* is small).Then, choose *B* to consist of a small set of small primes (with $p^1 = -1$) so that several b_i's are *B*-numbers when $b_i^2 (mod\ n)$. It is important to note that given the collection of vectors in F_2^h, we can be sure to find a subset of them that sum to zero because we are looking for a collection of vectors that are linearly dependent over the field. We know from linear algebra that $(\underset{B_1}{\rightarrow}, \ldots, \underset{B_h}{\rightarrow},)$ is linearly dependent in F_2^h, if we have *h* + 1 vectors that are *h*-tuples. So we will have to generate at most *h* + 1 distinct *B*-numbers to find when

$$\left(\prod_i b_i\right)^2 \equiv \left(\prod_j (p_j)^{\frac{1}{2}\sum_i \alpha_{ij}}\right)^2 \ (mod\ n). \quad (15)$$

If *h* is sufficiently large, it helps to write the vectors as rows in a matrix and use row-reduction techniques to find linearly dependent set of rows that sum to zero. [5][6]

We can now outline a systematic method of factoring *n* using Fermat's method and factor bases. First, choose an integer *y* of intermediate size (Koblitz gives the example of an intermediate size - if *n* is fifty digits, let *y* be five or six digits [6]). Let *B* be all of the primes less than or equal to *y* and −1. Next, choose a large number of random b_i and try to express b_i^2 (*mod n*) as least absolute residues that are products of the primes in *B*. Once we have obtained ($\pi(y)$ + 2) *B*-numbers (where $\pi(y)$ is the number of primes that do not exceed *y* [1]) we generate the corresponding vectors, , $\underset{B_i}{\rightarrow}$,), in F_2^h (where $h = \pi(y) + 1$) in order to get linear dependence. We then put the vectors into a matrix and use row-reduction to determine the subset of b_i's whose corresponding $\underset{B_i}{\rightarrow}$ add up to zero. We then let $b = \prod ib_i\ (mod\ n)$ and $c = \prod j(p_j)^{\frac{1}{2}\sum i\alpha ij}\ (mod\ n)$ so that $b^2 \equiv c^2$ (*mod* n) (which gives us Equation 15) . If $b \not\equiv \pm c$ (*mod n*), compute *g.c.d.*(*b* + *c*, *n*) to get a nontrivial factor of *n*. If $b \equiv \pm c$ (*mod n*), choose a different subset of rows in the matrix of $\underset{B_i}{\rightarrow}$'s that sum to zero, adding a few more *B*-numbers and their corresponding rows if necessary. If this does not work then we start the process over again with new random collection of *B*-numbers. [6]

Continued Fraction Method

The continued fraction method of factorization is a refinement of Fermat's method of factorization with factor bases. In the previous section we needed to find a reliable method of finding integers *b* where $1 \le b \le$ n such that the least absolute residue b^2 (*mod n*) is a product of small primes,

which is likely to happen when $|b^2 \pmod n|$ is small. This method uses continued fractions, hence the name, to find b's such that $|b^2 \pmod n| < 2\sqrt{n}$. So we need to understand a few things about continued fractions before we can understand this factorization method.

A continued fraction is defined as follows: Given a real number x, we construct its continued fraction by first letting

$$q_0 = \lfloor x \rfloor \text{ and set } x_0 = x - q_0. \text{ Then let } q_k = \left\lfloor \frac{1}{x_{k-1}} \right\rfloor \text{ and } x_k = \frac{1}{x_{k-1}} - q_k \text{ until } \frac{1}{x_{k-1}}$$

is an integer (and therefore $x_k = 0$). The process will terminate when x is rational because all x_k's will be rational with decreasing denominators. Therefore, the continued fraction looks like

$$x = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cdots \cfrac{1}{q_k + x_k}}} \tag{16}$$

$$= q_0 + \frac{1}{q_1+}\frac{1}{q_2+}\cdots\frac{1}{q_k + x_k} \tag{17}$$

where Equation 17 is the compact notation of Equation 16 and each continued fraction is unique by the way we have defined it [3][6]. So the general form of a continued fraction is

$$q_0 + \frac{1}{q_1+}\frac{1}{q_2+}\cdots\frac{1}{q_k} \tag{18}$$

and we can let $q_0, q_1, \ldots, q_k$ be variables in order to manipulate the continued fraction into the expression of a quotient of two sums where each sum is composed of various products from the set $\{q_0, q_1, \ldots, q_k\}$. We see that if $k = 1, 2, 3$ we can manipulate the continued fraction as

$$q_0 + \frac{1}{q_1} = \frac{q_0 q_1 + 1}{q_1} \tag{19}$$

$$q_0 + \frac{1}{q_1+}\frac{1}{q_2} = q_0 + \frac{q_2}{q_1 q_2 + 1} = \frac{q_0 q_1 q_2 + q_0 + q_2}{q_1 q_2 + 1} \tag{20}$$

$$q_0 + \frac{1}{q_1+}\frac{1}{q_2+}\frac{1}{q_3} = q_0 + \frac{q_2 q_3 + 1}{q_1 q_2 q_3 + q_1 + q_3} = \frac{q_0 q_1 q_2 q_3 + q_0 q_1 + q_0 q_3 + q_2 q_3 + 1}{q_1 q_2 q_3 + q_1 + q_3} \tag{21}$$

where the value of $q_1 + \frac{1}{q_2}$ in Equation 20 is derived from Equation 19 and evaluating q_1 and q_2 in place of q_0 and q_1 respectively. Equation 21 was derived in the same way using Equation 20. Thus, we can build out the general continued fraction in this manner and can represent the numerator of Equation 18 as $[q_0, q_1, \ldots, q_k]$. When we look at Equations 19, 20, and 21 we see that the denominator of the expression can be represented as $[q_1, q_2, \ldots, q_k]$ since the denominator is derived from the numerator of the previous answer. Therefore, we can represent the general continued fraction as

$$q_0 + \frac{1}{q_1+}\frac{1}{q_2+}\cdots\frac{1}{q_k} = \frac{[q_0, q_1, \ldots, q_k]}{[q_1, q_2, \ldots, q_k]}.$$

We can also see from Equations 19-21 how the numerator is derived as the recurrence relation

$$[q_0, q_1, \ldots, q_k] = q_0[q_1, q_2, \ldots, q_k] + [q_2, q_3, \ldots, q_k] \tag{22}$$

This will work for all k provided that when $k = 1$ we interpret the second bracket as 1 ($[q_0, q_1] = q_0[q_1] + 1 = q_0 q_1 + 1$, which is what it is supposed to be). [3]

So we see that $[q_0, q_1, \ldots, q_k]$ is the sum of certain products formed from the set $\{q_0, q_1, \ldots, q_k\}$, but how do we form the products? The answer was found by Euler, who was the first to give a general account of continued fractions, and created "Euler's rule" to form the products. The rule instructs us to first take the product of all the terms, then take every product that can be acquired by omitting any pair of consecutive terms, then take every product that can be acquired by omitting any two separate pairs of consecutive terms, and so on and so forth. The summation of these products is equal to the value of $[q_0, q_1, \ldots, q_k]$. If n is odd (giving us $n + 1$ terms, which is even) we add the empty product or the product of all the terms omitted, which we have defined as 1. This can be proved by induction using the recurrence relation in Equation 22. We assume the rule holds for the right hand side of Equation 22 and we have to prove that it holds for the left hand side. $[q_2, q_3, \ldots, q_k]$ is the sum of all the products formed from the set $\{q_0, q_1, \ldots, q_n\}$ where q_0 and q_1 have been omitted. Therefore $q_0[q_1, q_2, \ldots, q_k]$ is the sum of all the products formed from the set $\{q_0, q_1, \ldots, q_n\}$ where q_0 and q_1 is not omitted since all the products must contain q_0. When that factor is removed we are left with the sum of products from the set $\{q_1, \ldots, q_n\}$ where any separate pairs of consecutive terms are omitted. Thus, we get the appropriate sum of products and the rule holds for the function $[q_0, q_1, \ldots, q_k]$ for all k. [3]

Davenport points out that an immediate deduction of Euler's rule is that the value of $[q_0, q_1, \ldots, q_k]$ is the same if the terms are rewritten the reverse order, meaning $[q_0, q_1, \ldots, q_k] = [q_k, q_{k-1}, \ldots, q_0]$. This is true because we can express $[q_0, q_1, \ldots, q_k]$ in terms of a similar function as Equation 22, except with the last term or last two terms omitted:

$$[q_0, q_1, \ldots, q_k] = q_n[q_0, q_1, \ldots, q_{k-1}] + [q_0, q_1, \ldots, q_{k-2}] \tag{23}$$

This is equivalent to Equation 22 because we can rewrite the terms in the opposite order to get $[q_k, q_{k-1}, \ldots, q_0] = q_k[q_{k-1}, \ldots, q_0] + [q_{k-2}, \ldots, q_0]$ which is a restatement of Equation 22 with different symbols. [3] We can now define the i-th convergent of the continued fraction as the continued fraction terminated at qi where $i < k$ of

the general continued fraction (Equation 18). The value of the *i*-th convergent is

$$q_0 + \frac{1}{q_1+}\frac{1}{q_2+}\cdots\frac{1}{q_i} = \frac{[q_0, q_1, \ldots, q_i]}{[q_1, q_2, \ldots, q_i]} = \frac{A_i}{B_i},$$

where $A_i = [q_0, q_1, \ldots, q_i]$ and $B_i = [q_1, q_2, \ldots, q_i]$ to simplify notation. We see from this definition that the first convergent is $\frac{A_o}{B_o} = \frac{q_0}{1}$, the last convergent $\frac{A_k}{B_k}$ is the continued fraction itself, and A_i and B_i for $0 \le i \le k$ are natural numbers defined by the sums and products of q_i's as we have defined above. The recurrence relation from Equation 23 can now be represented as

$$A_i = q_i A_{i-1} + A_{i-2} \quad (24)$$

$$B_i = q_i B_{i-1} + B_{i-2} \quad (25)$$

where B_i has q_0 omitted. [3] From these two equations, we can show that any two consecutive convergents will differ by a factor of $(-1)^{i-1}$, or

$$A_i B_{i-1} - B_i A_{i-1} = (-1)^{i-1}, \quad (26)$$

which we can prove by induction. When $i = 1$, we have: $A_0 = q_0$, $B_0 = 1$, $A_1 = q_0 q_1 + 1$, and $B_1 = q_1$. So $A_1B_0 - B_1A_0 = (q_0q_1 + 1)(1) - (q_0)(q_1) = 1$. We can prove this generally by substituting Equations 24 and 25 into Equation 26 to get

$$\begin{aligned} A_iB_{i-1} - B_iA_{i-1} &= (q_iA_{i-1} + A_{i-2})B_{i-1} - (q_iB_{i-1} + B_{i-2})A_{i-1} \\ &= A_{i-2}B_{i-1} - B_{i-2}A_{i-1} \\ &= -(A_{i-1}B_{i-2} - B_{i-1}A_{i-2}). \end{aligned} \quad (27)$$

Therefore, the expression on the left of Equation 26, which we will call δ_i, has the property that $\delta_i = -\delta_{i-1}$ which can then be recursively defined to give us $\delta_i = -\delta_{i-1} = +\delta_{i-2} = \cdots = \pm\delta_i$ where $\pm\delta_i$ is +1 if *i* is odd and −1 if *i* is even. Therefore, it can be represented by $(-1)^{i-1}$ due to the fact that $\delta_1 = 1$ and Equation 27 is true. [3] A consequence of Equation 26 is that A_i and B_i will always be relatively prime and therefore the fraction B_i is in lowest terms, which means that all convergents (including the continued fraction itself) is in lowest terms. So when we represent a rational number $\frac{a}{b}$ as a continued fraction, the convergents of the continued fraction compose a sequence of rational numbers with the last one being $\frac{a}{b}$ We can see that the convergents will alternately be less than and greater than the final value of $\frac{a}{b}$ by rewriting Equation 26 as

$$\frac{A_i}{B_i} - \frac{A_{i-1}}{B_{i-1}} = \frac{(-1)^{i-1}}{B_iB_{i-1}}. \quad (28)$$

Furthermore, since $B_i < B_j$ for $0 \le i < j \le k$ the difference in Equation 28 will decrease as *i* increases. We can also see that the even convergents will be less than $\frac{a}{b}$ the odd convergents will be greater than $\frac{a}{b}$, and that each convergent will be closer to $\frac{a}{b}$ than the previous convergent. [3]

In order to see the continued fraction method of factoring, we need to use these facts to prove that $|b^2 \pmod n| < 2\sqrt{n}$. First, if $x > 1$ is a real number whose continued fraction expansion has convergents $\frac{b_i}{c_i}$, then $|b_i^2 - x^2c_i^2| < 2x$. Since *x* is between $\frac{b_i}{c_i}$ and $\frac{b_{i+1}}{c_i+1}$ and the absolute value of the difference between the successive convergents is $\frac{1}{c_ic_i+1}$ (from Equation 28) we can see that

$$\begin{aligned} |b_i^2 - x^2c_i^2| &= c_i^2\left|x - \frac{b_i}{c_i}\right|\left|x + \frac{b_i}{c_i}\right| \\ &< \frac{c_i^2}{c_ic_{i+1}}\left(x + \left(x + \frac{1}{c_ic_{i+1}}\right)\right). \end{aligned}$$

Thus,

$$\begin{aligned} |b_i^2 - x^2c_i^2| - 2x &< 2x\left(-1 + \frac{c_i}{c_{i+1}} + \frac{1}{2xc_{i+1}^2}\right) \\ &< 2x\left(-1 + \frac{c_i}{c_{i+1}} + \frac{1}{c_{i+1}}\right) \\ &< 2x\left(-1 + \frac{c_{i+1}}{c_{i+1}}\right) = 0. \end{aligned}$$

So if *n* is a positive integer which is not a perfect square, we can let $\frac{b_i}{c_i}$ be the convergents in the continued fraction expansion of $\sqrt{n}$ so that the residue of $b_i^2 \pmod n$ is the smallest in absolute value (between $-\frac{n}{2}$ and $\frac{n}{2}$) is less than $\sqrt[2]{n}$. When we apply the previous theorem with $x = \sqrt{n}$ we see that $b_i^2 \equiv b_i^2 - nc_1^2 \pmod n$ and the latter integer is less than $\sqrt[2]{n}$ in absolute value. This is the key to the continued fraction method of factorization because we can find a sequence of b_i's that have small residues when squared by taking the numerators of the convergents of the continued fraction expansion of $\sqrt{n}$. We do not even have to find the actual convergent since we are only using the numerator $b_i \pmod n$ and the fact that the numerator and denominator of the convergents become large is not a problem since we are working with integers less than or equal to n^2 when we are multiplying integers modulo *n*. [6]

So here is the algorithm for the continued fraction method of factoring. Let *n* be the integer that we want to find its factorization. We first set $b_{-1} = 1$, $b_0 = a_0 = = \lfloor\sqrt{n}\rfloor$ and $x_0 = \sqrt{n} - a_0$ and compute $b_0^2 \pmod n$ [or $b_0^2 - n$]. Next, for $i = 1, 2, \ldots$ we first set $a_i = \left\lfloor\frac{1}{x_{i-1}}\right\rfloor \pmod n$, then set $b_i = a_ib_i{-}1 + b_{i-2} \pmod n$ and compute $b_i^2 \pmod n$. After this is done for several *i*, we look at the *bi* (*mod n*) which factor into ± a product of small primes. We then create a factor base *B* that are

composed of $-1$and the primes that occur more than once of the set $b_i^2 \pmod n$ or occur with an even power in just one$b2 \quad 2i \pmod n$. We then list all of the numbers $b_i^2 \pmod n$ which are *B*-numbers, along with their correspondingβi of zeros and ones and find a subset whose vectors sum to zero. We then set $b = \prod ib_i \pmod n$ and $c = \prod p_j^{\frac{1}{2}\sum \alpha ij}$ where p_j are the elements of $B\backslash\{-1\}$ and the sum is taken over the same subset of *i*. If $B \not\equiv \pm c \pmod n$, then *g.c.d.*$(b + c, n)$ is a nontrivial factor of *n*. If $b \equiv \pm c \pmod n$, then we evaluate a different subset of *i* such that $\sum \overrightarrow{B_i} = 0$. If this is not possible, then more a_i, b_i, and $b_i^2 \pmod n$ must be computed to expand our factor base *B*. [6]

Which Method Is The Best?: The Advantages and Disadvantages of Each Method

While there are other methods of factoring that could be used (e.g. elliptic curve factorization, index calculus method, etc.) in order to crack RSA cryptography, these three methods are the simplest factoring methods that I could find. So now that we have these three methods, which one is the best for cracking RSA cryptography?

Pollard's Rho Method vs. Fermat's Method (Part I)

Pollard's rho method of factorization is the simplest factorization algorithm of the ones discussed in this paper. Simple calculations such as evaluating a chosen function, doing modular arithmetic, and finding the greatest common divisor of two numbers are all simple to do by hand or to code into a computer program (and all of these calculations are computationally inexpensive to calculate), yet there are some disadvantages to this method. The first one is that there are many computations that need to be done to find the factorization. This increase in the number of computations can lead to errors (when calculating by hand as well as coding a computer program) and increase the time it takes to find the answer. Even if the functions are computationally inexpensive to calculate in a computer program, the number of calculations that have to be done will add up and therefore take longer to find the prime factors. Second, we may not find the factor on the first attempt which means that we would have to start over with either a new function f, a new starting value x_0, a new way to compute the *g.c.d.*, or any combination of these three inputs. The third problem is that there are too many choices for inputs. It is not something that can immediately be executed [like Fermat's method (Part I)] and the choices that are made do not guarantee success, thus causing our choices to be re-evaluated for the second round of calculations if we do not get a proper factor.

Fermat's method (Part I), on the other hand, is a little more complex than the rho method in its computations, yet it is immediately executable and will give us the answer the first time (even if it takes a while). The disadvantage to Fermat's method is that when it is being executed by a computer program there is a chance that there is a problem with overflow errors (e.g. t^2 or n exceed the upper bounds of the computer's computational limits).

In order to examine each method further I wrote C programs for both methods. For each method I tested 24 numbers that were 19 digits in length (the highest my computer could test) where the primes were randomly generated and had varying distances between each other. The results for each of these numbers can be seen in Table 4 on page 15, but my findings can be summarized in Table 3 where I look at the average time each method took to calculate the prime factors as well as if the method is successful on this first try. There are two things that are important to note about the results. The first is that for the rho method I used $f(x) = x^2 + x + 1$, $x_0 = 2$, and I used the first method to calculate the *g.c.d.* that was outlined in the rho method section on page 6. The second thing is that for Fermat's method (Part I), I was only able to write a program using Equations 7 and 8 and not Equations 9 and 10 since I was using values of *n* close to the upper bound of my computer's limits and calculating *kn* in Equation 9 would have caused overflow errors. As Table 3 shows, the rho method took an average time of 26 minutes to compute an answer and 33% of the time it was able to find the correct prime factorization of *n*. This means that 66% of the time it did not give me an answer, but it could with different initial conditions. Conversely, Fermat's method took an average time of less than a tenth of a second and was always able to find the factorization of *n*. Therefore, when comparing these two methods of factorization, I have found that Fermat's method of factorization is faster and more reliable.

Rho Method Average Time	Rho Method Found Factors (%)	Fermat Method Average Time	Fermat Method Found Factors (%)
1583.546 sec. ≈ 26.392 min.	8/24 = 1/3 = 33.333%	<.147 seconds	24/24 = 100%

Table 3: Summary of Results

Fermat's Method (Part I) vs. Continued Fraction Method and Fermat's Method (Part II)

Since the continued fraction method of factorization is a refinement of Fermat's factorization with factor bases, there is no point in comparing these two methods. Instead, we shall compare Fermat's method (Part I) and the continued fraction method. We can see from the descriptions of the Fermat's method (Part II) and continued fraction method that the process is more intricate and complex than Fermat's method (Part I), yet the mathematics involved is still elementary. While I was able to write a C program for Fermat's method (Part I), I was not able to for Fermat's method (Part II) or the continued fraction method due to the complexity of the process and my introductory knowledge of C programming. Thus I was not able to compare the two methods as I did with Pollard's rho method and Fermat's method (Part I). Assuming that I had the knowledge to write such a program, the calculations involved can be programmed on a computer with single-precision floating point and it lends itself to parallel processing. The computers can even be linked by e-mail to complete the task. The disadvantage is that it is very memory-intensive so that you need a good deal of computing power, while Fermat's method (Part I) can be done on a calculator. Also, the continued fraction method works best for numbers that are roughly 300 bits long. The reason for this is that the continued fraction method requires that we find primes less than the number we are factoring, and primality testing and trial division of large numbers becomes increasingly difficult. Therefore, for numbers that are roughly 300 bits long the continued fraction method works as well as Fermat's method (Part I) but can find the results quicker. For numbers greater than 300 bits long, it can still find the answer but Fermat's method (Part I) will probably be quicker. [2][5][8]

Conclusion of Cracking RSA Cryptography

We have looked at three different methods of factorization in this paper, and while they all have their advantages and disadvantages, they are all capable of factoring the public key in RSA cryptography. So is RSA cryptography secure? Yes. But how can that be? The first reason why this is true is the size of the public key. When I was factoring numbers with the computer programs I wrote, the largest number my computer could handle was 9, 000, 000, 114, 000, 000, 361 which is 19 digits long and less than 64 bits. RSA keys are typically 1024-2048 bits long [9], meaning that the public key n is $2^{1024} - 1 \leq n \leq 2^{2048} - 1$. That means the public key is somewhere between 309-617 digits long which is outside the bounds of any personal computer and cannot be factored with any C programs of the factorization methods above. The second reason RSA cryptography is secure is that even if we had a computer that could factor these numbers, it would take an extremely long time. The primes chosen to create n follow a set of conditions (see page 5) such that factorization is made extremely difficult and maximizes the amount of time to find its factorization. Also, the people who use RSA cryptogrpahy switch their keys on a regular basis. So by the time we have found the factorization of their public key, they have started using a new key and our work has been for nothing (except the satisfaction that we now know the factorization of a very large number, but that is not worth much). Therefore, RSA cryptography is secure not because it is uncrackable, but the amount of time and effort that must go into to cracking it is not realistic or feasible.

A Computer Calculation Results

Number Tested	Rho Method Time (in seconds)	Prime Factors Found?	Fermat Method Time (in seconds)	Prime Factors Found?
10861930631686550 33	561.631	Yes	2.293	Yes
15283068041091496 07	56.908	No	0.015	Yes
17257896281067870 91	279.489	Yes	<0.001	Yes
23226068470411482 53	55.489	Yes	0.046	Yes
26219886412655611 67	261.658	Yes	0.062	Yes
29738831496114125 57	471.385	No	0.249	Yes
30621205113770013 67	267.836	Yes	<0.001	Yes
35499840336417421 01	1348.027	No	0.015	Yes
39900001398350012 21	393.666	Yes	<0.001	Yes
43953002306290030 21	1107.490	Yes	<0.001	Yes
45484001307728082 07	1205.193	Yes	<0.001	Yes
47459064680192266 37	3251.851	No	0.015	Yes
54856323519563182 31	3033.154	No	<0.001	Yes
57592221647925452 17	796.848	No	<0.001	Yes
59726734517575775 03	3542.182	No	0.592	Yes
60220745502887623 51	1622.228	No	<0.001	Yes
62108303784101001 53	2818.655	No	<0.001	Yes
67170451546778511 71	3004.887	No	<0.001	Yes
70470002709900025 97	317.569	No	0.062	Yes
74502652984359738 23	1616.971	No	0.015	Yes
77831958558460633 13	3938.641	No	<0.001	Yes
80166526816209538 01	5768.512	No	0.156	Yes
87936589590196463 51	2056.314	No	<0.001	Yes
89998502939981522 57	228.524	No	<0.001	Yes

Table 4: Results of Computer Program for Rho and Fermat Method

B Glossary

B-number - an integer b where the least absolute residue $b^2 \pmod n$ can be written as a product of numbers in the factor base B.
ciphertext - the encrypted message
cryptanalyst - someone who tries to break cryptographic systems without knowledge of the key.
cryptanalysis - the study of breaking cryptographic systems.
cryptographers - someone who tries to create unbeatable cryptographic systems.
cryptography - the study of methods of sending encrypted messages so that only the intended recipient can decrypt and read the message.
deciphering/decryption - process of converting ciphertext to plaintext.
enciphering/encryption - process of converting plaintext to ciphertext.
factor base - a set $B = \{p_1, p_2, \ldots, p_k\}$ of distinct primes (except p_1 may be -1).
key - the element that turns the general encryption algorithm into a specific method of encryption.
least absolute residue of $a \pmod n$ - is the integer in the interval $\left[\frac{-n}{2}, \frac{n}{2}\right]$ to which a is congruent.
plaintext - the original message to be sent.
private key - the private information used by the receiver to decrypt the message sent.
public key - the algorithm and public information that is used by the sender to encrypt the message.

recurrence relation - an equation that recursively defines a sequence; using a recursive definition to define an equation.
residue classes modulo n - the set of equivalence classes congruent modulo n, meaning that any integer is represented on the set $\{0, 1, \ldots, n-1\}$.

C Computer Code for Pollard's Rho Method

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

long long f(long long x, long long int n)
{
    long long ans;
    ans=(x*x+x+1)%n;
    return ans;
}

long long gcd(long long a, long long b)
{
    long long c;
    do
    {
        c=a%b;
        a=b;
        b=c;
    }while(c!=0);
    return a;
}

int main(void)
{
    unsigned long long n, x=2, y, z, w, p, q;
    int counter=0, h=1, k=0;
    printf("What is n?\tn= %llu",n);
    scanf("%llu", &n);
    printf("\n");
    clock_t start = clock();
    do
    {
        counter++;
        if(k<counter)
            do
            {
                h=2*h;
                k=h - 1;
            }while(k<counter);
        y=f(x,n);
        w=abs(y-z);
        p=gcd(w, n);
        if(k==counter)
        {
            z=y;
        }
        x=y;
    }while(p==1);
    q=n/p;
    printf("\nTime elapsed: %f seconds\n", ((double)clock() - start) / CLOCKS_PER_SEC);
    printf("\nThe factors of %llu are %llu and %llu.\n\n", n, q, p);
    return 0;
}
```

D Computer Code for Fermat's Factorization (Part I)

```
#include<stdio.h>
#include<math.h>
#include<time.h>
int main(void)
{
    unsigned long long int n, p, q, t, s1, x;
    long double n2, sroot, sroot2, s;
    int ctr=0;
    printf("What number do you wish to factor?n = ");
    scanf("%llu", &n);
    printf("%llu", n);
    clock_t start = clock();
    n2=(long double)n;
    sroot = sqrt(n2);
    sroot2 = floor(sroot);
    t = (unsigned long long int) sroot2;
    do{
        t++;
        x = (t*t)-n;
        s=sqrt((long double) x);
        ctr++;
    }while(floor(s)!=s);
    s1 = (unsigned long long int) s;
    p=t+s1;
    q=t-s1;
    printf("\nTime elapsed: %f seconds\n", ((double)clock() - start) / CLOCKS_PER_SEC);
    printf("\nThe factors of %llu are %llu and %llu.\n\n", n, q, p);
    return 0;
}
```

E Picture Credits

Figure 1 taken from http://www.thedavincigame.com/Boardgame/Frequency_Analysis.png
Figure 2 taken from http://enigma.wikispaces.com/file/view/Vigenere-square.png/30615996/Vigenere-square.png
Figure 3 taken from http://www.mathmu.cn/doc/images/rho.png

References

[1] G.E. Andrews, *Number Theory*, Dover Publications, Inc., New York, 1971.

[2] R. Crandall, C. Pomerance, *Prime Numbers: A Computational Perspective*, Springer, New York, 2005.

[3] H. Davenport, *The Higher Arithmetic: An Introducion to the Theory of Numbers*, Cambridge University Press, Cambridge, 2008.

[4] W. Diffie, M. Hellman, New Directions in Cryptography, *IEEE Trans. Inf. Theory, IT-22*, 1976, 644-654.

[5] J. Katz, Y. Lindell, *Introduction to Modern Cryptography*, Chapman & Hall/CRC, New York, 2008.

[6] N. Koblitz, A Course in Number Theory and Cryptogrpahy, Springer-Verlag, New York, 1987.

[7] N. Koblitz, A. Menezes, The Brave New World of Bodacious Assumptions in Cryptography, *Notices of the AMS*, 57(3), 2010, 357-365.

[8] I. Niven, H.S. Zuckerman, H.L. Montgomery, *An Introduction to the Theory of Numbers*, John Wiley & Sons, Inc., New York, 1960.

[9] *RSA Cryptography* available at <http://en.wikipedia.org/wiki/RSA >.

[10] S. Singh, *The Code Book: The Science of Secrecy From Ancient Egypt to Quantum Cryptography*, Anchor Books, New York, 1999.