RSA Encryption and the Pursuit of Large Primes

Bryant Prieur

2 June 2014

1 Introduction

For a pretty long time, people have occasionally wanted to pass messages in private - that is, in such a way that only the intended recipient will be able to read the message. Unfortunately for those such people, there exists another class of people that takes great interest in the communications of others. So cryptography was born.

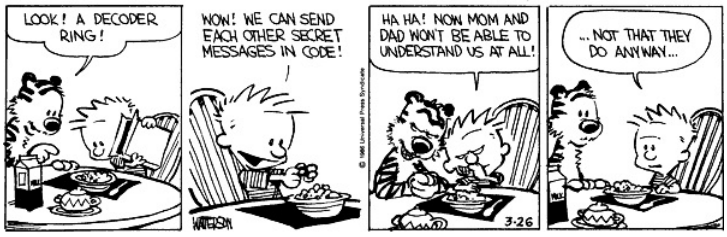

Figure 1: Motivations for Cryptography (Watterson)

The invention of the internet transitioned a great deal of communication over to digital means such as email. This was, no doubt, tremendously exciting for nosy people who understood the technology as a fair bunch of these communications were completely unencrypted and relatively easy to obtain. As such, several methods of encrypting these communications were invented, among them: RSA Encryption.

2 RSA Encryption

The "RSA" in RSA Encryption stands for Rivest-Shamir-Adleman, the surnames of those who patented[1] the method in 1977. It belongs to the family of "public-key cryptosystems," systems that rely on the existence of a public key - well known and used to encrypt information - and a private key - known only to the intended recipient and used to decrypt data.

The keys are mathematically linked, but it is difficult to obtain the private key from the public. Essentially, these systems are "hard to crack, but easy to check." So that if you already had the private key, you could simply decipher the message. If you did not, you would be very sad as you frittered your life away trying to.

With the general notion of the goals of the RSA Encryption method in mind, let us examine the method itself. Before any mathetmatical encryption can take place, any string of characters meant to be encrypted must be transformed into a string of integers. This is a simple exercise in character mapping - from one character to a string of digits. Once a recipient has decoded their message to a string of digits, he or she need only apply the mapping in reverse to obtain the original message.

The mathematical encryption relies on two prime integers, let us say p, q. From here, we define $n = pq$, $k = (p-1)(q-1)$ and d such that d and k are relatively prime (that is, their greatest common factor is 1). Finally, let us also choose e such that $de \equiv 1(\mod k)$

Given these properties, it is true that $b^{ed} \equiv b(\mod n)$ for every $b \in \mathbb{Z}$. This is the heart of the encryption method. Taking the string of digits from earlier, we split it into blocks such that the resulting integer in each block is less than n. Let us call our block b.

The encrypter calculates the remainder of b^e/n, let us call this c and sends this as the encrypted message. The recipient then calculates the remainder of c^d/n and receives b. What mathematical wizardy is this? It is merely an application of the property that $b^{ed} \equiv b(\mod n)$ for every $b \in \mathbb{Z}$.

Our first calculation for c is simply $b^e \equiv c(\mod n)$. In our second, we find $c^d \equiv b(\mod n)$ which is true because $c^d \equiv (b^e)^d = b^{ed} \equiv b(\mod n)$. We have essentially used this property to go from b^e to b^{ed} which is congruent

[1] Rivest, Ronald L., Adi Shamir, and Leonard M. Adleman. Cryptographic Communications System and Method. Massachusetts Institute of Technology, assignee. Patent US 4405829 A. 14 Dec. 1977. Print.

to b. Up until now you have taken my word for it, but let us show this is true:

We know that $de \equiv 1(\mod k)$, which is to say $de - 1 = kt$ for some integer t. Then we can rearrange that to say $de = kt + 1$. This gives us:

$$b^{ed} = b^{kt+1} = b^{kt}b = b^{(p-1)(q-1)t}b = (b^{(p-1)})^{(q-1)t}b$$

Fermat's Little Theorem then allows us to take this final result and say:

$$(b^{(p-1)})^{(q-1)t}b \equiv (1)^{(q-1)t}b \equiv b(\mod p)$$

The public portion of the system are the numbers n and e. With these, anyone can perform an encryption. However, one cannot traverse the congruency from b to b^{ed} without also knowing d. The number d is essentially the private portion of the system. (You would also have to keep k secret as given k, you can calculate d. This means keeping p, q secret as well, as they would give you k. Of course, none of these values are actually used in the decryption process so it may make the best sense to forget them.)

Someone attempting to crack the RSA system given only the public portion of the system would therefore have to factor n into p and q. It is therefore advisable to make n very difficult to factor - a value of $n = 15$ for instance could be cracked by a 3rd grader. As such, we turn to the power of computing to generate very large primes for use in RSA cryptography.

3 Generating Large Primes

3.1 The Sieve of Eratosthenes

A rather old method for generating primes, the Sieve of Eratosthenes finds all primes less than or equal to n. To do so, we list the integers from 2 to n. Every time we reach some number we haven't crossed out, let us say x, we consider it a prime and proceed jump x numbers ahead and cross out the number we land on until we reach the end. Then we return to where we left off and repeat.

Intuitively, this is easy to understand. We begin with 2, and cross of its multiples as they are clearly composite numbers. The next number we haven't crossed of must be prime, so we cross off its multiples, so on and so forth. Every number we reach that is not a multiple of those before it must therefore be prime.

This is an effective method, but somewhat slow. Assuming we implement it as described, we must generate the list, giving us n operations, and then go through it for every number we reach. At worst, we have $O(n!)$.

A better implementation checks values from the list as it is generated, giving us $O(n)$. If we desire truly large primes, we will have to do better.

3.2 The Sieve of Sundaram

This sieve was created in 1934 by P. Sundaram. This finds all odd prime integers from 3 to $2n + 2$. Starting from a list from 1 to n, it removes all integers of the form $i + j + 2ij$ where $i, j \in \mathbb{Z}^+$ and $1 \leq i \leq j$. All remaining values are doubled and incremented by 1, giving us the primes between 3 and $2n + 2$. (Of course, it cannot figure out that 2 is a prime, but you and I know it is.)

To show this works, let us consider an excluded integer. It takes the form $i + j + 2ij$, and let us apply the operation that would turn an integer not of this form into a prime:

$$2(i + j + 2ij) + 1$$

$$2i + 2j + 4ij + 1$$

$$(2i + 1)(2j + 1)$$

Factoring the second line gives us the third, revealing that nontrivial factors. Therefore, we have only received odd integers with trivial factors - i.e. primes. As such, this sieve works at $O(2n)$. It is slightly faster than the Sieve of Eratosthenes, and certainly we could leave it running for a few hours to generate some truly large prime integers.

4 Generating Very Large Primes

So far, we have obtained a lot of primes at once. However, things could go faster if, instead, we managed to generate one value at a time and find out it is a prime integer. One such way we might do this is through primality testing.

4.1 Primality Testing

4.1.1 Fermat Primality Test

Fermat's Little Theorem states that for a prime integer p and $1 \leq a \leq p$, that $a^{p-1} \equiv 1(\mod p)$. Therefore, a simple way to test the primality of some supposed prime q is to choose an a such that $1 \leq a \leq q$ and see if $a^{q-1} \equiv 1(\mod q)$. If not, clearly q is not a prime. However, if the statement holds, then q might be a prime.

In a computational setting, we may randomly select values for a and check. The more values we select, the better chance there is of having obtained a prime. (Unfortunately, the realm of mathematics is an evil place and there exist such things as Carmichael Numbers[2] that, despite being composite, pass the Fermat Primality Test for every a. Note this does not violate Fermat's Little Theorem. My libraries include an implementation of the Miller-Rabin Primality Test which weeds out Carmichael Numbers 3/4 times.)

4.2 Generating Large Numbers that Might be Primes

Given that we can test numbers for primality, it may simplify our life to try to select numbers for testing that may already be prime. One such way to do so is to select from the Mersenne Numbers. The n^{th} Mersenne Number is defined as:

$$M_n = 2^n - 1$$

Clearly, these are all odd integers. This is convenient, as only one even integer is prime. Let us now rule out a few Mersenne Numbers as composite. Let us consider $a, b \in \mathbb{Z}$. We may have:

$$M_{ab} = 2^{ab} - 1$$

Unfortunately, this is factorable:

$$M_{ab} = 2^{ab} - 1 = (2^a - 1)(1 + 2^a + 2^{2a} + ... + 2(b-1)a)$$

[2]Carmichael, R. D. "Note on a New Number Theory Function." Bulletin of the American Mathematical Society 45.4 (1939): 269-74. Web.

From this, we know that all composite values of n result in composite Mersenne Numbers. But therefore, we may use small prime values (such as generated by our earlier sieves) to create much larger numbers that have a fair chance of being prime. We may then test their primality to be sure. Given all this, we have generated large primes that will have a difficult-to-factor product for use in RSA cryptography.