Introduction to Cryptography and RSA

Prepared by Leonid Grinberg for 6.045 (as taught by Professor Scott Aaronson)

Spring 2011

1 The basics of cryptography

Cryptography is the practice and science of securing information. This document will discuss a particular cryptographic method (really a family of cryptographic methods) that can be used to transfer infromation securely and conveniently between two parties. We will talk both about the theory and practice of doing this and the various other nice things that come as a result.

1.1 Symmetric cryptography

Arguably the simplest cryptographic method is something called "one-time pad." Consider as an example two imaginary characters, Alice and Bob (you'll see these guys cropping up a lot in cryptographic texts), who want to communicate in secret with one another. Let's say in particular that Alice has some secret message that she wants to send to Bob. For simplicty we're going to assume that the message is written in a binary format (we can use some encoding scheme to convert it to binary if it isn't). She and Bob meet up beforehand and generate a long secret "key," which is a random string of `0`s and `1`s. Let's say her message is "`01101`" and the key is "`10110`." Then, to "encrypt" her message, Alice generates a new string (sometimes called the "cyphertext" to differentiate between the unencrypted form, called the "plaintext") which has a "`0`" if the corresponding bit in the original message and the key are the same, and a "`1`" otherwise (the reader may recognize this as the "XOR" function). So in our example, the cyphertext would be "`11011`." She sends this to Bob, which he can then decrypt by applying the same operation (XOR done twice is the null operation—try it!)

This scheme works, but it has some problems. The biggest issue is that the key has to be as long as the message itself, and Alice and Bob need to be able to meet up beforehand in secret and agree on it. If they are able to do this, then it raises the question of why they couldn't have just sent the original message to begin with!

Now, Alice and Bob can cheat a little and generate a much shorter secret key, and then just keep going back to the beginning when they run out of bits. So if the key was 5 bits long as in the previous example and the message was much longer, then the encrypting process would involve copying the key over and over (so in the case of a 10-bit message like "`1101100111`," the key "`10110`" would be applied first to the "`11011`" part and then to the "`00111`" part to produce "`0110110001`."

The problem with this is is that statistical analysis can be used to notice patterns that may arise as the result of any structure in the message. For instance, the distribution of letters and combinations of letters in English is extremely non-uniform, with some appearing far more frequently than others. Since reusing the key will preserve some of these patterns, a computer may be able to use the information to get back some or even all of the message.

1.2 Pseudorandom number generators

Another solution uses something called "*pseudorandom number generators*" (PRNGs). A PRNG is a type of function that takes a small amount of random bits (called the "*seed*") and uses that to generate a new sequence of bits that appear random, but are generated deterministically based on the seed. Then, Alice and Bob need only to agree on a much shorter seed instead of on the whole key. As long as the seed is kept secret, they can keep generating the same "random" numbers and use one-time pad with them to encrypt and decrypt messages that they send to one another.

The problem with this system is that no one knows if PRNGs actually exist! There are some candidates that people use in practice, but no one can prove that they actually are hard to break. And, once again, the system does still require agreeing on a seed, which may not always be possible.

2 A new approach: Asymmetric cryptography

When using one-time pad and the variant with PRNGs, we were using a system that is inherently "symmetric"—that is, the same key is used to encrypt and decrypt data, and each party, Alice and Bob, could do either operation (and, in fact, this operation turned out to be the same because of how binary XOR works).

What if we had a system where that wasn't the case, where the process of encrypting something is entirely separate from the process of decrypting it? Would we gain something from using such a system?

2.1 A physical analogy

As a motivation for the system we are about to explore, consider the following analogy that uses physical objects:

Alice wants to send Bob some contraband. Alice and Bob live in a totalitarian society where all mail is intercepted by the government. However, Alice and Bob each have a unique lock for which only they have the respective keys. The locks are unbreakable and if the government can't open a locked package, it will just forward it to the recipient.

How can Alice get the contraband to Bob? She can put it in a box and lock the box with her lock. She sends the locked box to Bob. If the government intercepts the package, they can't open it because they don't have Alice's key. When Bob gets the package, he can't open it either because he also doesn't have Alice's key, but he can put his own lock on the box alongside Alice's. He sends the box back, and again, the government can't open it. When Alice gets it, she takes off her lock (but can't open the box because Bob's lock is still on it) and sends the box back again. This time, when Bob gets it, he can take off his lock and open the box to get the map out.

Now, in the world of cryptography, we will be using mathematical functions, not physical locks. So for our purposes, there can exist multiple copies of the "lock," which Bob can send to whomever wants to send him a message, thus making this whole procedure even simpler—Bob just sends Alice a copy of his lock, and she applies it to the message and sends it back to him (his key is still kept secret and once she locks the message she can't unlock it, but neither can anyone else besides Bob).[1]

3 Getting to the actual math

Enough beating around the bush. How can we actually do this? The algorithm we're about to explore is a slightly simplified version of something called RSA, named after the initials of its inventors—Rivest, Shamir, and Adleman—who discovered it in the 1970s at MIT.

3.1 A quick note on terminology

Borrowing from how the word was used in traditional symmetric cryptographic schemes like one-time pad, the word "key" is used to mean both the operation that encrypts a message and decrypts it. Rather than splitting it as "lock" and "key," from now on, we will use "*public key*" and "*private key*," respectively. For this reason, this type of scheme is often known as a "*public key cryptographic system*" (and in general, this whole field is often referred to as "*public key cryptography*").

3.2 Outline of the procedure

Alice wants to send Bob some message (for simplicity, we will again assume that it's a binary string). She indicates to him that she wishes to do so, and he will generate a public/private key pair and send the public key to her. She will use the key to encrypt her message and send the result back to him. He will then use the private key that only he knows to decrypt the message.

3.3 Bob's keys

When picking an operation to apply to the keys, we are looking for something that should be relatively easy to compute in one direction but hard in the other. There are several examples of such operations in mathematics, but the one RSA uses is prime factorization.

Recall that a prime number is one that is divisible only by 1 and itself. Given two prime numbers, it is easy to multiply them—you can just use long multiplication, and the total number of operations will be about quadratic in the total number of digits of the two primes. However, given some number that is the product of two primes, getting those two primes back can be quite difficult. In fact, so far no one has discovered any method of doing so

[1]In fact, one can imagine a directory where everyone's lock were publicly available, and to send them a message, you would just look up their public lock and encrypt the message. Such systems do in fact exist.

that is fundamentally better than brute force, which takes exponential time in terms of the number of digits of the original number.

Going off this fact, Bob will randomly generate two prime numbers as his private key.[2] We'll call these primes p and q. These numbers have to be quite large: about 250 bits is a standard amount which translates to something like 60 digits. He will then multiply them together, and send the product (we'll call it N) as his public key to Alice.

3.4 Alice's encryption process

Let's call Alice's message x. To encrypt her message, Alice cubes it and sends the result modulo N. For technical reasons, it is important that $x < N < x^3$. This can be accomplished by padding x with random "junk" data that will be ignored after decryption.

After Alice sends this, any eavesdropper will have so far only seen N (from when Bob sent it) and $x^3 \mod N$ (from when Alice sent it); as far as anyone knows, these are insufficient to determine the value of x in any reasonable amount of time.

3.5 An intermission

Before we can talk about how Bob goes about decrypting the message, we need to build up a bit more mathematical background. We'll start by defining a relatively simple function called $\varphi(N)$, which is defined as the number of positive integers less than N that are relatively prime to N (recall that two numbers are relatively prime if their greatest common divisor is 1). So, for instance $\varphi(15) = 8$, since the following positive integers are less than 15 and relatively prime to it:

1 2 4 7 8 11 13 14

3.5.1 Analyzing this set of numbers

Let's take a look at these numbers. Note that everything we're about to observe applies in the general case to any number that is the product of two primes, not just 15. However, for clarity and simplicity, we'll focus on these numbers as our examples.

There are a few things that are worth noticing about this set of numbers. The first is that 1 will always be present in it, as every number is relatively prime to 1. The second is that multiplication mod N (in this case $N = 15$) is an operation that this set is closed under (in other words, if a and b are two numbers in this set, then $ab \mod N$ will be as well—this is easy to verify).

Another important note, and one that we will not prove here, is that for any number, there is precisely one number in the set that is its "*inverse*" mod N—that is, for some number a in the set, there exists precisely one number b in the set such that $ab \equiv 1 \mod N$.

[2]It is an important question of how he actually goes about doing this. As it turns out, the process of checking *whether* a number is prime is quite easy (as compared to actually getting the prime factors), and the probability that an n-digit number is prime is about $\frac{1}{2.3n}$. Thus, Bob can just randomly guess numbers until he finds two that are prime.

3.5.2 Cycles

Let's pick an arbitrary number from the set that is not 1—in our example, we'll go with 2. Keep multiplying it by itself mod N. We get 2, 4, 8, 1 (16 mod 15), and then back to 2 again—a cycle. If we had picked another number that is not in that cycle, like 7, and multiplied it by itself, we would have gotten 7, 4 (49 mod 15), 13 (28 mod 15), 1 (91 mod 15), and back to 7 again—a new cycle.

The fact that we get cycles of some form is not surprising. Since there is a finite amount of positive integers that are relatively prime and less than N, we will clearly eventually have to repeat a number. Since each number has a unique inverse, this will mean that we have completed a cycle.

In this case, though, the cycles are only subsets of the total set. The question is, how big are these subsets?

Let's take the first cycle again—2, 4, 8, 1. Let's multiply each of these (mod N of course) by some other number not in the cycle—say, 11. We get 7 (22 mod 15), 14 (44 mod 15), 13 (88 mod 15), and 11. Now, let's see what would have happened if we had multiplied it by another number, say, 7. We would have gotten 14, 13 (28 mod 15), 11 (56 mod 15), and 7. Notice that both of these cycles are permutations of one another.

Now, in this particular case, there is no number we can multiply this cycle by to get anything but a permutation of this set or of the original cycle. If N were larger, this would not be the case. But there is no N for which it would be possible to take a cycle and multiply each of its elements by two different numbers and get back two different sets that are neither equal nor disjoint (i.e. that are not equal but still have some elements in common). This comes from the fact that all the numbers have unique inverses—you have to be able to "work backwards" from a set to get the original cycle, which must be unique.

3.5.3 Dividing up the set

So what does this tell us about the size of these cycles? What it gives us is what's called an "*equivalence relation*," which is just a fancy way of saying that we can group all these different sets of numbers into different categories. A given cycle, and all the sets that you can get from it by multiplying it by some number not in the original cycle comprise one such category. Each such category covers the entire set of numbers. Since each of the elements in the category is of the same size, it must be the case that the size divides the total amount of numbers we started with. Indeed, in our example, this is the case—each cycle is of length 4, and the entire set has 8 numbers in it.

3.5.4 Looping around and around

One thing we notice is that if we pick any number a and keep cycling around by multiplying by a repeatedly, the last number in the cycle before we get back to a will always be 1. Since the total size of the cycle divides the size of the whole set, it follows that if we loop around exactly the number of times as there are elements in the set, we will land on 1. In other words,

$$a^{\varphi(N)} \equiv 1 \mod N$$

for any positive integer a relatively prime to N.

3.5.5 Bringing it all together

The last question is, what is $\varphi(N)$? This can be determined via a simple counting argument. If $N = pq$, then there are a total of pq numbers between 1 and N (inclusive). Of those, p are divisible by q, and q are divisible by p, so we have to subtract those if we only care about those that are relatively prime to pq. But we subtracted one number (pq itself) twice, so we have to add 1 back to compensate. Thus,

$$\begin{aligned}\varphi(N) &= \varphi(pq)\\ &= pq - p - q + 1\\ &= (p-1)(q-1)\end{aligned}$$

and plugging back in, we get an important result that we're going to need:[3]

$$\begin{aligned}a^{(p-1)(q-1)} &\equiv 1 \mod N\\ a^{(p-1)(q-1)} &\equiv 1 \mod pq\end{aligned}$$

3.6 Bob's decryption process

Going back to Alice and Bob, we still have Bob left with the message that Alice encrypted: $x^3 \mod N$. Armed with Euler's Theorem, however, Bob can get back the value of x using his knowledge of p and q.

First, he chooses a number k such that $3k \equiv 1 \mod (p-1)(q-1)$. Such a number is not hard to find. What this means is that there exists some number c such that $c(p-1)(q-1) + 1 = 3k$. Bob then raises Alice's encrypted message to the power of k, remembering that the laws of modular arithmetic allow the modulo operation to go "through" multiplication:

$$\begin{aligned}(x^3 \mod N)^k &= x^{3k} \mod N\\ &= x^{c(p-1)(q-1)+1} \mod N\\ &= x \cdot (x^{(p-1)(q-1)})^c \mod N\\ &= x \cdot 1^c \mod N\\ &= x \mod N\end{aligned}$$

Thus, Bob gets back Alice's original message.

[3]This result actually has a name, "*Euler's Theorem,*" after Leonhard Euler (1707-1783). It can be proved in a variety of ways and is a central theorem of number theory.

4 Benefits and drawbacks

This system has the advantage over one-time pad that it does not require Alice and Bob to have any shared secrets before the algorithm is used. This is hugely advantageous, and is what allows real-world systems to use this system—for instance, when you visit a website over HTTPS, it first goes through this dance to encrypt the transmission.

There are some drawbacks, however. The entire security of the system depends on the difficulty of factoring numbers—for which there is no known fast algorithm, but which no one has yet to prove is intractable. On quantum computers, furthermore, there *does* exist a fast factoring algorithm (called Shor's algorithm after MIT professor Peter Shor who invented it).

On the other hand, full one-time pad (without PRNGs and without repeating the key) is *fundamentally unbreakable* because *any* message of a given length can be generated, assuming all possible values of the key are randomly and universally distributed.

In real-world deployments of RSA, furthermore, a lot of factors need to be taken into account with public key systems. The generation of the prime numbers, for instance, is key, as they need to be big enough and close enough together that the product is actually difficult to factor. It is also theoretically possible for a malicious observer to gain some amount of knowledge about the values of p and q by observing how long it takes to generate them, so sometimes systems will pad out that amount as well.

Lastly, the algorithm is not particualrly fast (at least when compared to one-time-pad), which makes it inconvenient for rapid back-and-forth communication. Therefore, often in real-world deployments, RSA will be used for mutual agreement of a seed, after which the systems will fall back to PRNG-backed one-time pad.

5 Bonus points: Authentication

Thus far, we have only been talking about encryption—that is, how Alice can send a message to Bob so that others can't read it. There is a related but separate problem of "*authentication*"—how can Alice prove that she is the one sending a message?

Public key cryptography gets us this for free! If Alice wants to send a message to Bob and prove that it is really she who is sending it, she can generate a public/private key pair, and use her private key on her message (which breaks down the physical analogy introduced earlier as she would in effect be unlocking an already unlocked box). She can then send it to Bob along with her public key, which he can apply to the message to get the original message back. He knows that the original message must indeed be coming from Alice, because only she could know the private key that matches the public key she sent him.

The asymmetry of this system means that it is easy to combine encryption and authentication. Alice simply applies this method to her message before encrypting it with Bob's public key. Bob would then first decrypt the message using his private key, and then apply Alice's key to authenticate it.