Cryptography

By Jeff's son Joshua Zachariah for grade 11

Email, the internet, online banking, online documents, and the like allow lots of confidential information to be transferred. To this end encryption has been invented. The two main uses of encryption are identifying oneself and sending messages. Instead of looking at many applications of cryptography, this report will outline the basic math used in the most common encryption method, RSA. Before doing this, we will have to learn about finite fields and before learning about finite fields, we will have to consider the extended GCD algorithm.

The goal of Cryptography is the following. I choose a private key that I keep secret and a public key I share with the world. Someone wanting to send a message to me uses the public key to encode his message. I am able to decode the message because I know the private key. Anyone intercepting the massage, but not knowing this private key is unable to learn anything useful. In the case of identification I can prove who I am by decoding a message encoded with the public key by using the private key. Your password is effectively this private key.

All encryption methods are based on what is known as one-way-functions, that is functions y=f(x) such that computing y from x is easy but learning x from y is computationally very hard. RSA, for example is based on the fact that multiplying two primes p and q together to give N is easy, but factoring n to retrieve p and q is very hard. Of course not impossible because one could simply check whether 2, 3, 4, 5,… sqrt(N) divides into N. However, if N is a 100 digit number than multiplying p and q would only take 100log(100) = 700 operations while factoring would take about sqrt(10^{100}) which is more than the number of atoms in the universe..

Extended GCD Algorithm

The standard GCD algorithm finds the greatest common factor of two numbers. For example GCD(6,4)=2. The algorithm uses the fact that GCD(a,b) = GCD(b,a%b). Hence, replacing a with b and b with a%b makes our numbers smaller while maintaining the fact that GCD(a,b) is the answer we want. The base case is that if b=0 then the GCD(a,b)=a. For example, GCD(76,64) = GCD(64,12) = GCD(12,4) = GCD(4,0) = 0. In the extended, GCD algorithm we get input (a,b) and get output (s,t,g) so that sa+tb=g.

Finite Fields

A field is like another Universe where the laws of math work slightly different. For example in a universe adding could work like this: $a + b \equiv (a + b)\%p$. The rules of a field are not arbitrary though and fall under certain restrictions.

Key rules for a field: A universe of objects U and operations +,-,*, and / defined on them.

- There is a zero such that a+zero = a and a*zero = zero
- There is a one such that a*one = a
- Given a, there is an additive inverse b such that a+b = zero. We write b = -a.
- Given a, there is a multiplicative inverse b such that a*b=1. We write b = a^{-1}.

- Communicative: a+b=b+a
- Associative: a+(b+c) = (a+b)+c
- Distributive: a*(b+c) = a*b + a*c

A Finite Field is what its name implies, it's a field that has a set size and we can't go beyond that.Our Finite "Universe" will be U= {0, 1, 2, … p-1} where p is a prime number. The advantages to such a system is that because this is a finite set, you can represent a number accurately unlike reals. Finite Fields also don't "overflow" (they don't get to big). And finally a finite field can divide unlike an integer without the possibility of reals. In our Universe we need to define +, -, ×, ÷ and to this we use Mod. In our Universe $a + b \equiv a + b \ \% \ p$ and $a \times b \equiv a \times b \ \% \ p$.

One issue that can be thought about is how to compute $a^b \equiv a^b \ \% \ p$ quickly when a^b is too big to fit into our computer. . How we go about this is we have an algorithm that says

```
Power (a, b) {
        If (b = 0)
                Return 1
        Else
                r = power (a, b/2)
                If (b is even)
                        Return (r^2 % p)
                Else
                        Return (r^2 × a % p)
}
```

How this works is that is that we know that $a^b = a^{b/2}a^{b/2}$ = r^2. This doesn't work though if b is odd because we don't want to use reals. So what we do is we round b/2 down and by doing this loose an accuracy of one. To remedy, this we say $a^b = a^{b/2}a^{b/2}a$ so that we gain back the one we lost. Therefore if b is odd we return $(r^2 \times a)$.

Another thing to think about how to find the inverse a^{-1} of a given element: a. Dividing $1/a$ will not work because we want to stay away from reals. We require that, $a^{-1} \equiv s$ such that $a \times s \equiv 1 \ \% \ p$. For example $2 \times s \equiv 1 \ \% \ 7$ where s=4. To find *s*, we use the extended GCD algorithm (see above) which states that when given (a,p) it will return (s,q,g) such that $as + pq = g$ = GCD(a,p). In our case p is prime so the g=GCD(a,p)=1 (unless $a \equiv 0 \ \% \ p$). Because as plus some integer number of p's equals one, it follows that $a \times s \equiv 1 \ \% \ p$ as required.

In conclusion, the integers mod p forms a finite field. In contrast, the integers mod n is not a field if n=pq. The easiest way to see this is that p is not zero and q is not zero but $p \times q \equiv 0 \ \% \ n$. These are called zero divisors and cause havoc on our math universe. This means that every number does not have an inverse mod n. But some of them do. Suppose GCD(a,n) = 1. It may be that neither a nor n are prime, but with respect to each other they are what we call co-prime. In this case, we can still find $a^{-1} \equiv s$ such that $a \times s \equiv 1 \ \% \ n$. Again the extended GCD algorithm returns (s,q,g) such that $as + pn = g$ = GCD(a,p) 1. Because as plus some integer number of n's equals one, it follows that $a \times s \equiv 1 \ \% \ n$ as required.

Fermat's little theorem

$m^{(p-1)} = 1 \bmod p$
if p is a prime

The intuition is that if one starts with 1 and keeps multiplying by m, you get different numbers 1, m, m^2, m^3 … All of these need to be in our universe (excluding zero) {1,2,3, …, p-1}. There are only p-1 of these so you must repeat. After p-1 of them you get back to 1, giving (as required) that $m^{p-1} = 1 \bmod p$. A stranger version that we will need for RSA is that $m^{(p-1)(q-1)} = 1 \bmod pq$.

RSA

RSA is based on the fact that prime factoring is very hard to compute. The standard players are called Alice, Bob, and Eve. Alice chooses two prime numbers p and q. From these she computes n=pq, e (for encode) and d for (decode). The public key <n,e> she tells the world. The private key <n,d> she keeps secret. When Bob wants to send a message m to Alice but he doesn't want the message to be seen so he encodes the message Code=Encode (m, <n,e>) using Alice's public key. Alice, can use her private key to decode it, *m=Decode (Code, <n,d>).* Eve (eavesdropper) knows the public key <n,e> and learns the code by eavesdropping. However, the belief is that she can't learn anything useful about Bob's message m without effectively factoring n=pq which requires far too much computation time than she has.

Let us now do the math in more detail. Alice can find to 100 digit primes p and q randomly choosing integers and testing to see if they are prime. Then she computes n=pq and $\Psi = (p-1)(q-1)$. She chooses some e for which $\mathrm{GCD}(e, \Psi) = 1$. Recall that $\mathrm{GCD}(e, \Psi) = 1$ allows Alice to use the Extended GCD algorithm (above) to find d such that $de = 1 - t\Psi \equiv 1 \ \%\Psi$. As said, she makes <n,e> public and she keeps <n,d> private. Bob encodes the message (m) with c=Encode (m, <n,e>) $= m^e \% n$ and sends her c. When Alice decodes the message with *m=Decode (Code, <n,d>)* $= c^d \% n$. What is required now is proving that Alice does in fact accurately recover the message m.

What she decodes is

$m' = c^d = (m^d)^e = m^{de}$ (because $de \equiv 1 \% \Psi$ and $de = 1 + t\Psi$)

$= m^{1+t\Psi} \% n$

$= m \times (m^{t\Psi})^t \% n$ (Recall Fermat's little theorem (see above) $m^{(p-1)(q-1)} \equiv 1 \bmod pq$.

Also recall that $\Psi = (p-1)(q-1)$ and $n = pq$)

$= m \times 1^t \% n = m \% n$

The third and final part of this story is that if Eve knows n,e, and c and can find m then through some algorithm (I don't know what it is) she can find p and q. but because of contra positive if she can't find p and q (which we believe she can't) she can't find m.

Conclusion

The computer age has brought lots of freedoms but more freedoms means leaving us open to more security breaks. Luckily computers are able to quickly do the mathematics we need so that we can secure this given freedom without being quick enough to allow Eve to break this security.