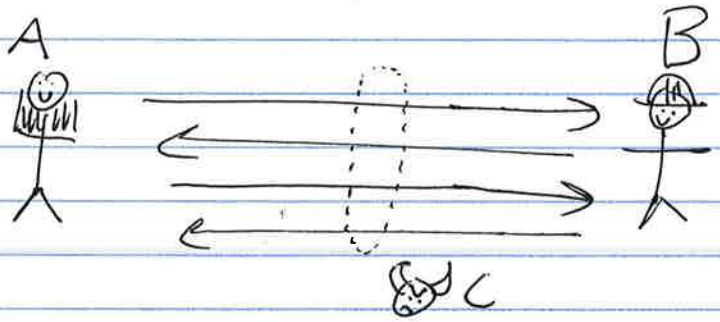
1

3510 L4 Cryptography

Cryptography is the science of secrecy. We learn about it in the class (algorithms) because it is all about algorithms for secure communication. Consider the following scenario. You have two parties we denote as Alice, Bob. They may communicate and pass messages to each other. There is a third party, the adversary, perhaps denoted Charlie. Charlie can listen to what Alice and Bob send. Is there an algorithm, a protocol in which Alice and Bob could agree to perform to stop Charlie from listening.>



This is not such a contrived scenario. You and Amazon could be parties A, B. The protocol would stop someone from stealing your credit card info. Much of the financial sector relies on cryptography. What is a protocol to solve this problem.

Here is one of the first ones invented called one time pad. Suppose that Alice wants to send Bob a message m of length $|m| = n$. Alice and Bob already have a secret key $|s| = n$. Alice computes $m \oplus s$ and sends it to Bob. Bob computes $(m \oplus s) \oplus s = m$ and learns the message. Charlie learns only $(m \oplus s)$ by eavesdropping. If s is very random, then Charlie can learn nothing from looking at $(m \oplus s)$.

This works and is fast but has three immediate issues.

1 How do A, B already know the secret s.? They can't send it or C would find it out.

2 s is as long as the message! We want small keys and long messages. We want to send a lot of info and not store gigabytes of keys.

3 It's called one time pad for a reason. Suppose Bob, after learning m, wants to reply with a message m', so he sends $m' \oplus s$. So Charlie now has $m \oplus s$ and $m' \oplus s$. He may compute $m \oplus m'$ which provides information to the adversary!

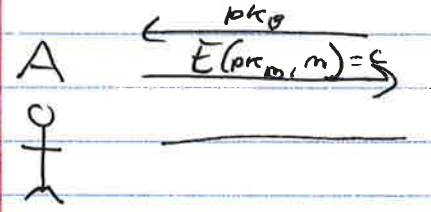
There are three directions in cryptography to fix this. First is to use a symmetric key encryption algorithm. $E_K(m) = c$ and $D_K(c) = m$. Here K can be much shorter than m, and given c it is computationally infeasible to compute m from c. Another way is to use asymmetric cryptography. An asymmetric cryptosystem is a pair of 3 algorithms G, E, D where

$(sk, pk) \leftarrow G(1^n)$

$E(pk, m) = c$

$D(sk, E(pk, m)) = m.$

~~Decr~~ encryption is done with the public key but decryption is done with a secret key. First we discuss the protocol and then the definition of security.



Bob sends his public key pk_B to Alice, who encrypts her message m as $E(pk_B, m)$. She sends this to Bob who decrypts it with the secret key sk. The adversary eavesdrops and learns $pk_B, E(pk_B, m)$. From just this information, we want to ensure they learn nothing.

How can we give a definition of security? nonexistence of certain algorithms. There always exists one attack against any system, guessing the key. $Pr[C \text{ guesses } K] = 2^{-n}$ where $|K| = n$. This could take exponential time. We may intuitively say that a protocol is secure if the best attack is guessing the key. We say then the protocol is secure if $Pr[C \text{ learns anything about } m] < 2^{-n}$.

3 gcd.

Now we have described an idealized asymmetric cryptosystem. lets actually give a protocol for it. First we need to develop facts from number theory.

$\gcd(a,b)$ computes the greatest common divisor of a, b. For example $\gcd(3\cdot 5, 2\cdot 3) = 3$. There is an easy divide and conquer algorithm discovered long ago by Euclid: $\gcd(a,b) = \gcd(b, a \bmod b)$ If $b=0$ return a. We can prove correctness by showing $\gcd(a,b) = \gcd(b, a-b)$. Repeated application gets us the mod. Let $d = \gcd(a,b)$. Then $d \mid a$, $d \mid b$, so $a = dk$, $b = d\ell$. so $a - b = dk - d\ell = d(k-\ell)$ so d is a factor of $a-b$, so $d \mid a-b$. so $d \mid \gcd(b, a-b)$. let $\gcd(b, a-b) = d'$. So $d' \mid b$, $d' \mid (a-b)$. So $d' \mid (a-b)+b = a$. so $d' \mid a$ and $d' \mid b \Rightarrow d' \mid \gcd(a,b) \Rightarrow d' = d$. Easy!

What is the runtime? Since $a \bmod b < a/2$, every two recursive calls shrinks our answer by one bit. $2n$ calls, each performing an n bit remainder in time $O(n^2)$, total takes $O(n^3)$.

We can use the stack trace execution of the euclidean algorithm to compute modular inverse. The inverse of an element a is some a^{-1} so that $a a^{-1} \equiv 1 \bmod N$. For example, $3^{-1} = 2 \pmod 5$ since $3 \cdot 2 = 6 \bmod 5 = 1$. Similarly, $2^{-1} = 3 \bmod 5$. Unlike real numbers where every nonzero element a has inverse a. Only sometimes do elements have inverses. $\boxed{a^{-1} \text{ exists } (\bmod N) \iff \gcd(a,N) = 1.}$ If a is relatively prime to N, then it has an inverse. If N is prime, then all elements less than it have inverse.

4

Fermats Little Theorem.

We prove the following insanely useful fact. If p is prime and $1 \le a < p$ then

$$a^{p-1} \equiv 1 \pmod{p}.$$

This is called fermats little theorem. Let $S = \{1, 2, \ldots p-1\}$ and $aS = \{a, 2a, \ldots (p-1)a \bmod p\}$. First we show (mod p) that $S = aS$. It is sufficient for us to show none of ~~0~~ & aS and elements of aS are distinct. Suppose aS contains two equal elements. $ai \equiv aj \pmod{p}$ Since $\gcd(a,p) = 1$ (p prime) ~~we may divide to get $i = j$, a contradiction~~. multiply by a^{-1}. $a^{-1}ai \equiv a^{-1}aj \Rightarrow i \equiv j$, a contradiction. Now suppose $ai \equiv 0 \bmod p$. then $a^{-1}ai \equiv a^{-1}0 \pmod{p} \Rightarrow i \equiv 0 \bmod p$, but $i \ge 1$, contradiction.

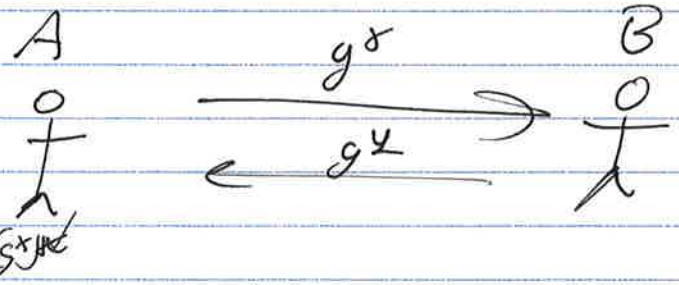
Since $aS = S$, we may product the elements of both sides. TTS $= (p-1)!$ $1 \cdot 2 \cdot 3 \cdots p-1 = (p-1)!$ product of $aS = a \cdot (2a) \cdot (3a) \cdots a(p-1) \equiv a^{p-1}(p-1)!$ So $(p-1)! \equiv a^{p-1}(p-1)!$ Since $\gcd((p-1)!, p) \equiv 1$, then we see $a^{p-1} \equiv 1 \pmod{p}$.

We state a generalization we wont prove. $\varphi(N)$ = eulers totient function. $\varphi(N)$ = the number of numbers less than N but relatively prime to N. If p is prime, $\varphi(p) = (p-1)$. If $N = pq$, $p \neq q$, then $\varphi(N) = (p-1)(q-1)$. Eulers theorem is a generalization of Fermats: If $\gcd(a, N) = 1$ then $a^{\varphi(N)} \equiv 1 \pmod{N}$. we dont have the tools to prove it, but it should be clear that FLT is a special case of ET.

We believe, but cannot prove the following are computationally infeasible:
Factoring: Given $N = pq$, determine p, q.
RSA: given y, $x^y \pmod{N}$ and N determine x.

All asymmetric crypto relies on unproven assumptions that we only have strong evidence for. The security is, unfortunately conditional. To prove security, you assume these problems are hard.

I will leave you with one more protocol, Diff Hellman key exchange.
Let p, g be chosen so that $\{g, g^2, g^3, \ldots g^{p-1}\} \bmod p$ is a reordering of $\{1, 2, 3, \ldots (p-1)\}$. We use the assumption that DH: given g^x, g^y, p, it is infeasible to learn $g^{xy} \pmod p$



Alice randomly generates x and computes and sends g^x. Bob randomly generates y and computes and sends g^y. Then they individually compute $(g^x)^y$ and $(g^y)^x$. Since these are equal, they now have the same value, g^{xy}. The adversary learns g^x, g^y, p but can never learn g^{xy}. The exponent is hidden.

This often used to share a secret key, and then apply symmetric cryptography, as it is much much faster.

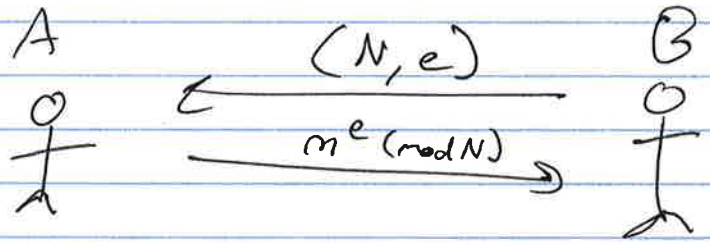
5 RSA

We finally have enough tools to describe the RSA protocol:

1 Bob generates large primes p, q
2 Bob computes $N = pq$
3 Bob picks some e relatively prime to $\varphi(N) = (p-1)(q-1)$
4 Bob computes $e^{-1} = d \pmod{\varphi(N)}$
5 Bob broadcasts $(N, e) = pk$, keeps $d = sk$ a secret
6 Alice computes $m^e \pmod N$ and sends/returns it to Bob
7 Bob computes $(m^e)^d \pmod N$ to get m. Why does this work?

Note that since $ed \equiv 1 \pmod{\varphi(N)}$, then $ed = k\varphi(N) + 1$ so $(m^e)^d \equiv (m^{ed}) \equiv m^{k\varphi(N)+1} \equiv (m^k)^{\varphi(N)} m \equiv (m^{\varphi(N)})^k m \equiv 1^k m \equiv m$. So $(m^e)^d \equiv m \pmod N$

Why is this secure? The adversary learns what?



$N, e, m^e \pmod N$. By our RSA assumption, it is infeasible to infer any information about m from just this.

Note if the adversary could factor N, they could compute $\varphi(N) = (p-1)(q-1)$ and then $d \equiv e^{-1}$. So they could learn d, if they could factor n and then learn $(m^e)^d \equiv m$.