CS 124 **Lecture 17**

17.1 Cryptography Fundamentals

Cryptography is concerned with the following scenario: two people, Alice and Bob, wish to communicate privately in the presence of an eavesdropper, Eve. In particular, suppose Alice wants to send Bob a message x. (For convenience, we will always assume our message has been converted into a bit string.) Using cryptography, Alice would compute a function $e(x)$, the encoding of x, using some secret key, and transmit $e(x)$ to Bob. Bob receives $e(x)$, and using his own secret key, would compute a function $d(e(x)) = x$. The function d provides the decoding of the encoding $e(x)$. Eve is presumably unable to recover x from $e(x)$ because she does not have the key – without the key, computing x is either impossible or computationally difficult.

17.1.1 One-Time Pad

A classical cryptographic method is the *one-time pad*. A one-time pad is a random string of bits r, equal in length to the message x, that Alice and Bob share and is secret. By random, here we mean that r is equally like to be any bit string of the right length, $|r|$. Alice compute $e(x) = x \oplus r$; Bob computes $d(e(x)) = e(x) \oplus r = x \oplus r \oplus r = x$.

The claim is that Eve gets absolutely no information about the message by seeing $e(x)$. More concretely, we claim

$$\Pr(\text{message is } x \mid e(x)) = \Pr(\text{message is } x);$$

that is, knowing $e(x)$ gives no more information to Eve than she already had. This is a nice exercise in conditional probabilities.

Since $e(x)$ provides no information, the one-time pad is completely secure. (Notice that this does not rely on notions of computational difficulty; Eve really obtains no additional information!) There are, however, crucial drawbacks.

- The key r has to be as long as x.
- The key r can only be used once. (To see this, suppose we use the same key r to encode x and y. The Eve can compute $e(x) \oplus e(y) = x \oplus y$, which might yield useful information!)

- The key r has to be exchanged, by some other means. (Private courier?)

17.1.2 DES

The *Data Encrytpion Standard*, or DES, is a U.S. government sponsored cryptographic method proposed in 1976. It uses a 56 bit key, again shared by Alice and Bob, and it encodes blocks of 64 bits using a complicated sequence of bit operations.

Many have suspected that the government engineered the DES standard, so that they could break it easily, but nobody has shown a simpler method for breaking DES other than trying the 2^{56} possible keys. These days, however, trying even this large number of keys can be accomplished in just a few days with specialized hardware. Hence DES is widely considered no longer secure.

17.1.3 RSA

RSA (named after its inventors, Ron Rivest, Adi Shamir, and Len Adleman) was developed around the same time as DES. RSA is an example of *public key cryptography*. In public key cryptography, Bob has two keys: a public key, k_e, known to everyone, and a private key, k_d, known only to Bob. If Alice (or anyone else) wants to send a message x to Bob, she encrypts it as $e(x)$ using the public key; Bob then decrypts it using his private key. For this to be secure, the private key must be hard to compute from the public key, and similarly $e(x)$ must be hard to compute from x.

The RSA algorithm depends on some number theory and simple algorithms, which we will consider before describing RSA. We will then describe how RSA is *efficient* and *secure*.

17.2 Tools for RSA

17.2.1 Primality

For the time being, we will assume that it is possible to generate large prime numbers. In fact, there are simple and efficient *randomized algorithms* for generating large primes, that we will consider later in the course.

17.2.2 Euclid's Greatest Common Divisor Algorithm

Definition: The *greatest common divisor* (or gcd) of integers $a, b \geq 0$ is the largest integer $d \geq 0$ such that $d|a$ and $d|b$, where $d|a$ denotes that d divides a.

Example: gcd(360,84) = 12.

One way of computing the gcd is to factor the two numbers, and find the common prime factors (with the right multiplicity). Factoring, however, is a problem for which we do not have general efficient algorithms.

The following algorithm, due to Euclid, avoids factoring. Assume $a \geq b \geq 0$.

```
function Euclid(a,b)
if b = 0 return(a)
return(Euclid(b, a mod b))
end Euclid
```

Euclid's algorithm relies on the fact that $\gcd(a,b) = \gcd(b, a \bmod b)$. You should prove this as an exercise.

We need to check that this algorithm is efficient. We will assume that mod operations are efficient (in fact they can be done in $O(\log^2 a)$ bit operations). How many mod operations must be performed?

To analyze this, we notice that in the recursive calls of Euclid's algorithms, the numbers always get smaller. For the algorithm to be efficient, we'd like to have only about $O(\log a)$ recursive calls. This will require the numbers to shrink by a constant factor after a constant number of rounds. In fact, we can show that the larger number shrinks by a factor of 2 every 2 rounds.

Claim 1: $a \bmod b \leq a/2$.

Proof: The claim is trivially true if $b \leq a/2$. If $b > a/2$, then $a \bmod b = a - b \leq a/2$. ■

Claim 2: On calling Euclid(a,b), after the second recursive call Euclid(a',b') has $a' \leq a/2$.

Proof: For the second recursive call, we will have $a' = a \bmod b$. ■

17.2.3 Extended Euclid's Algorithm

Euclid's algorithm can be extended to give not just the greatest common divisor $d = \gcd(a,b)$, but also two integers x and y such that $ax + by = d$. This will prove useful to us subsequently, as we will explain.

Extended-Euclid(a,b)

if $b = 0$ return$(a,1,0)$

Compute k such that $a = bk + (a \bmod b)$

(d,x,y) = Extended-Euclid$(b, a \bmod b)$

return$((d,y,x-ky))$

end Extended-Euclid

Claim 3: The Extended Euclid's algorithm returns the correct answer.

Proof: By induction on $a+b$. It clearly works if $b = 0$. (Note the understanding that all numbers divide 0!) If $b \neq 0$, then we may assume the recursive call provides the correct answer by induction, as $a \bmod b < a$. Hence we have x and y such that $bx + (a \bmod b)y = d$. But $(a \bmod b) = a - bk$, and hence by substitution we get $bx + (a - bk)y = d$, or $ay + b(x - ky) = d$. This shows the algorithm provides the correct output. ■

Note that the Extended Euclid's algorithm is clearly efficient, as it requires only a few extra arithmetic operations per recursive call over Euclid's algorithm.

The Extended Euclid's algorithm is useful if we wish to compute the inverse of a number. That is, suppose we wish to find $a^{-1} \bmod n$. The number a has a multiplicative inverse modulo n if and only if the gcd of a and n is 1. Moreover, the Extended Euclid's algorithm gives us that number. Since in this case computing $\gcd(a,n)$ gives x,y such that $ax + ny = 1$, we have that $x = a^{-1} \bmod n$.

17.2.4 Exponentiation

Suppose we have to compute $x^y \bmod z$, for integers x,y,z. Multiplying x by itself y times is one possibility, but it is too slow. A more efficient approach is to repeatedly square from x, to get $x^2 \bmod z$, $x^4 \bmod z$, $x^8 \bmod z \ldots$, $x^{2^{\lfloor \log y \rfloor}} \bmod z$. Now x^y can be computed by multiplying together modulo z the powers that correspond to ones in the binary representation of y.

17.3 The RSA Protocol

To create a public key, Bob finds two large primes, p and q, of roughly the same size. (Large should be a few hundred decimal digits. Recently, with a lot of work, 768-bit RSA has been broken; this corresponds to $n = pq$ being 768

bits long.) Bob computes $n = pq$, and also computes a random integer e, such that $\gcd((p-1)(q-1), e) = 1$. (An alternative to choosing e randomly often used in practice is to choose $e = 3$, in which case p and q cannot equal 1 modulo 3.)

The pair (n, e) is Bob's public key, which he announces to the world. Bob's private key is $d = e^{-1} \bmod (p-1)(q-1)$, which can be computed by Euclid's algorithm. More specifically, (p, q, d) is Bob's private key.

Suppose Alice wants to send a message to Bob. We think of the message as being a number x from the range $[1, n]$. (If the message is too big to be represented by a number this small, it must be broken up into pieces; for example, the message could be broken into bit strings of length $\lfloor \log n \rfloor$.) To encode the message, Alice computes and sends to Bob

$$e(x) = x^e \bmod n.$$

Upon receipt, Bob computes

$$d(e(x)) = (e(x))^d \bmod n.$$

To show that this operation decodes correctly, we must prove:

Claim 4: $d(e(x)) = x$.

Proof: We use the steps:

$$e(x)^d = x^{de} = x^{1+k(p-1)(q-1)} = x \bmod n.$$

The first equation recalls the definition of $e(x)$. The second uses the fact that $d = e^{-1} \bmod (p-1)(q-1)$, and hence $de = 1 + k(p-1)(q-1)$ for some integer k. The last equality is much less trivial. It will help us to have the following lemma:

Claim 5: (Fermat's Little Theorem) If p is prime, then for $a \neq 0 \bmod p$, we have $a^{p-1} = 1 \bmod p$.

Proof: Look at the numbers $1, 2, \ldots, p-1$. Suppose we multiply them all by a modulo p, to get $a \cdot 1 \bmod p, a \cdot 2 \bmod p, \ldots, a \cdot (p-1) \bmod p$. We claim that the two sets of numbers are the same! This is because every pair of numbers in the second group is different; this follows since if $a \cdot i = a \cdot j \bmod p$, then by multiplying by a^{-1}, we must have $i = j \bmod p$. But if all the numbers in the second group are different modulo p, since none of them are 0, they must just be $1, 2, \ldots, p-1$. (To get a feel for this, take an example: when $p = 7$ and $a = 5$, multiplying a by the numbers $\{1,2,3,4,5,6\}$ yields $\{5,3,1,6,4,2\}$.)

From the above equality of sets of numbers, we conclude

$$1 \cdot 2 \cdots (p-1) = (a \cdot 1) \cdot (a \cdot 2) \cdots (a \cdot (p-1)) \bmod p.$$

Multiplying both sides by $1^{-1}, 2^{-1}, \ldots, (p-1)^{-1}$ we have

$$1 = a^{p-1} \bmod p.$$

This proves Claim 5. ■

We now return to the end of Claim 4, where we must prove

$$x^{1+k(p-1)(q-1)} = x \bmod n.$$

We first claim that $x^{1+k(p-1)(q-1)} = x \bmod p$. This is clearly true if $x = 0 \bmod p$. If $x \neq 0 \bmod p$, then by Fermat's Little Theorem, $x^{(p-1)} = 1 \bmod p$, and hence $x^{k(p-1)(q-1)} = 1 \bmod p$, from which we have $x^{1+k(p-1)(q-1)} = x \bmod p$. by the same argument we also have $x^{1+k(p-1)(q-1)} = x \bmod q$. But if a number is equal to x both modulo p and modulo q, it is equal to x modulo $n = p \cdot q$. Hence $x^{1+k(p-1)(q-1)} = x \bmod n$, and Claim 4 is proven. ■

We have shown that the RSA protocol allows for correct encoding and decoding. We also should be convinced it is efficient, since it requires only operations that we know to be efficient, such as Euclid's algorithm and modular exponentiation. One thing we have not yet asked is why the scheme is secure. That is, why can't the eavesdropper Eve recover the message x also?

The answer, unfortunately, is that there is no proof that Eve cannot compute x efficiently from $e(x)$. There is simply a belief that this is a hard problem. It is an unproven assumption that there is no efficient algorithm for computing x from $e(x)$. There is the real but unlikely possibility that someone out there can read all messages sent using RSA!

Let us seek some idea of why RSA is believed to be secure. If Eve obtains $e(x) = x^e \bmod n$, what can she do? She could try all possible values of x to try to find the correct one; this clearly takes too long. Or she could try to factor n and compute d. Factoring, however, is a widely known and well studied problem, and nobody has come up with a polynomial time algorithm for the problem. In fact, it is widely believed that no such algorithm exists.

It would be nice if we could make some sort of guarantee. For example, suppose that breaking RSA allowed us to factor n. Then we could say that RSA is as hard as factoring. Unfortunately, this is not the case either. It is possible that RSA could be broken without providing a general factoring algorithm, although it seems that any natural approach for breaking RSA would also provide a way to factor n.