§8.3 Digital signatures.

RSA (in fact, public key cryptography in general) has an added bonus: We are able to easily verify the identity of the sender.

Suppose, as usual, Alice and Bob are communicating via RSA. Alice wants to send the message "m" to Bob (m an integer), but Bob wants to be sure the message is really coming from Alice (and that nobody is trying to trick him).

Recall that Bob has a public and private key:

Bob's public key: (e_B, n_B) ← B for Bob
Bob's private key: (d_B, n_B)

and Alice also has a public and private key:

Alice's public key: (e_A, n_A)
Alice's private key: (d_A, n_A) ← A for Alice.

Alice sends a signed message to Bob as follows:

(1) First, she computes $m^{e_B} \bmod n_B$, this is her usual encrypted message that she sends to Bob.

(2) Next, she computes $m^{d_A} \bmod n_A$, and sends this to Bob along with the encrypted message. What she is essentially doing is encrypting m with her own private key!

Bob receives the pair $(m^{d_A} \bmod n_A, m^{e_B} \bmod n_B)$ from Alice. He decrypts the encrypted message using his private key as usual:

$$(m^{e_B})^{d_B} = m \bmod n_B$$

(again, we have not yet proved why this works)

Bob now verifies that the message really came from Alice as follows:

1. Bob looks up Alice's public key (e_A, n_A)
2. Bob takes the first component of the message sent to him by Alice and computes:

$$(m^{d_A})^{e_A} \bmod n_A$$

$$\| $$

$$(m^{e_A})^{d_A} \bmod n_A$$

we know this is just "m" because of how decryption with RSA works.

3. If Bob gets "m" when he raises $m^{d_A} \bmod n_A$ to the power of Alice's public key, he knows the message came from Alice. Only she could know her private key, and thus provide a number whose power e_A would give m.

Example: We saw a detailed computation last week of the following situation:

Bob had keys: public $(13, 77)$
private $(37, 77)$.

Alice encrypted the message $m = 17$ to send to Bob by computing $17^{13} \bmod 77 = 73 \bmod 77$. Then she sent him the number "77".

Bob recovered the message by computing

$$77^{37} = 17 \bmod 77.$$

Now we add a digital signature from Alice to the picture. Suppose Alice chooses primes

$$p = 13, \quad q = 11, \quad \text{so} \quad n = 143 \quad \text{and} \quad \varphi(n) = 12 \cdot 10 = 120.$$

Then she chooses $e_A = 47$ so that her public key is $(47, 143)$ and her private key is found by solving

$$47 d_A = 1 \bmod 143$$

$$\Rightarrow d_A = 23, \quad \text{so it's} \quad (23, 143).$$

Then, in addition to sending Bob the encrypted message $m^{e_B} = 17^{13} = 73 \bmod 77$, Alice also sends

$$m^{d_A} = 17^{23} = 62 \bmod 143.$$

Bob already knows the message is $m = 17$. But he checks it is really from Alice by taking her public

key $e_A = 47$ and $n = 143$, and computing

$$62^{47} = 17 \bmod 143.$$

Since he gets 17, he knows it was really Alice that sent the message.

§8.4 Hash functions

First, let's explain the problem that Hash functions are going to solve.

For practical purposes, RSA encounters a problem: you can only encrypt and send one number (several hundred digits long). We want to be able to send longer messages, so the idea for doing so was to use RSA to send someone the key for a message you've encrypted using a secure symmetric encryption system.

As a "baby example" we saw how Alice could send Bob a long message encrypted using an LFSR to generate a keystream, and then use RSA to send him the seed for the LFSR.

But how do digital signatures work in a setting like this? How can they be used in more complicated settings?

First: We can still use signatures to allow Bob to verify who sent the message. Alice would send Bob the seed to the LFSR, encrypted using Bob's public key and Alice's private key. If they match when Bob decrypts them, he knows Alice is the sender.

Second: We would like to use digital signatures in the same way we use real signatures — to sign contracts! With short messages, we already have something that works: Suppose Bob wants Alice to agree to the contract:

"I owe Bob $100."

We could convert this into a binary number, say using ASCII, then change that into some decimal number "m". Alice could "sign" the contract by computing $m^{d_A} \bmod n_A$. (d_A ← Alice's private key) Then if Alice backs out, Bob can prove to authorities that Alice signed the contract by computing

$$(m^{d_A})^{e_A} \bmod n_A.$$

which will give m; the number m corresponds (via ASCII encoding and binary) to the message

"I owe Bob $100".

The authorities see this, realize that only Alice could have computed m^{d_A}, and arrest her for fraud.

Suppose, however, that the contract is very long, and so we cannot encrypt it using RSA. If we tried the LFSR-encryption trick + digital signatures, here is what could go wrong.

Bob types up an enormous contract for Alice to sign. He sends it to her. She seeds her favourite LFSR with a seed "s", and encrypts the contract. Then she sends the encrypted contract to Bob, along with the numbers $(s^{d_A} \bmod n_A, s^{e_B} \bmod n_B)$. Using this, Bob can certainly prove to authorities that Alice did sign the contract: He can prove that the seed "s" came from her, and show how applying "s" to the encrypted contract decrypts it.

!! HOWEVER !!

If Bob is dishonest, he can write a completely new contract, encrypt it using the seed "s", and then claim that this new contract was the thing Alice signed! This works because Alice only signed the seed for the LFSR, not the actual contract itself.

This is where hash functions come in. Suppose we have a function

$$f: \{\text{messages}\} \longrightarrow \text{natural numbers},$$

ie. a function that takes a message and gives back a number. This is what we did for short contracts; we did:

$$\text{message} \longmapsto \text{ASCII (thought of as a big binary number)}$$

and this was great because each message became a different number, ie. it was 1-1.

But now we just want f to be any reasonable function (ie don't send everything to zero, or something similar), f does not need to be 1-1.

Then if Alice wants to "sign" a contract "C", she can encrypt $f(C)$ using her private key, and this serves as her signature. Bob can't change the contract once she signs, because it might (or will if f is a complicated function) change the value of $f(C)$. (I.e if Bob makes a new contract C', it could be impossibly hard for him to rig his new contract so that $f(C) = f(C')$).

Let's do an example with a concrete function f to see this in action.

Example:

Suppose that f_n: {messages} $\longrightarrow$ integers is the function that converts every letter to an integer mod 26, then sums all those numbers mod n.

I.e. f_{50}(hi class)

$= 7+8+2+11+0+18+18 \mod 50$

$= 64 \mod 50$

$= 14 \mod 50.$

Suppose Alice and Bob decide upon using the hash function f_{50}. Bob sends Alice the contract

C = "Alice owes Bob one hundred dollars."

then apply f_{50} to this message gives 21 mod 50, since the sum is 271. For Alice to return the signed contract to Bob, she does the following:

First, if the contract is secret they exchange LFSR seeds and an encrypted contract as already described. However, Alice also now sends $f(C)$, encrypted with her private key. I.e. if her keys are

public = $(47, 143) = (e_A, n_A)$

private = $(23, 143) = (d_A, n_A)$

as in the last example, she sends

$f(C)^{d_A} = 21^{23} \mod 143 = 109 \mod 143$ as her "signature" for the contract.

If Bob tries to claim the contract actually said:

C' = "Alice owes Bob one thousand dollars"

then the authorities would investigate his claim by applying f_{50} to this new message/contract C', and finding $f_{50}(C') = 48$. $(298 \bmod 50)$

Then they would decrypt Alice's signature of 109 by computing

$$\begin{aligned} 109^{e_A} &\bmod 143 \\ &= 109^{47} \bmod 143 \\ &= 21 \bmod 143. \end{aligned}$$

Seeing that Alice signed a contract C with $f(C) = 21$, they know that Bob must've altered the contract and he is arrested for fraud.

Remark: It is possible for Bob to get clever and alter the contract to produce C' with $f_{50}(C) = f_{50}(C')$. This means we should have perhaps used a trickier hashing function. In real-world applications, the functions are so complicated that nobody has ever found C and C' with

$$f(C) = f(C') \quad \text{(where } f \text{ is the hash function)}.$$

However, we know such C and C' must exist (for pigeonhole principle reasons).

§8.5 Diffie-Hellman Key exchange.

As we saw earlier: If we're trying to send long messages, typically we would just use RSA to send a secret key by public channels. However, if that's all we want to do, then things can be easily streamlined.

The streamlined key exchange process based on modular exponentiation is called the Diffie-Hellman key exchange. It eliminates the need for parties to prepare public/private keys for communicating a secret k.

Diffie-Hellman key exchange: (Alice and Bob, as usual)

In this protocol, the key will be some number "k" modulo a prime p. They choose and share k as follows:

(1) Alice sends a large prime p to Bob and a second number q with $1 < q < p$. Alice chooses a large number A with $1 < A < p$ and also sends $q^A \bmod p$ to Bob, she keeps A a secret.

(2) Bob chooses a large B with $1 < B < p$, and sends $q^B \bmod p$ back to Alice, he keeps B secret.

(3) Alice computes $(q^B)^A \bmod p$ (she got $q^B \bmod p$ from Bob and kept A a secret), and Bob computes $(q^A)^B \bmod p$ (he got $q^A \bmod p$ from Alice and kept B a secret). Now both Alice and Bob know a common key $K = (q^B)^A = (q^A)^B \bmod p$.

Remark: All communication in the above scheme is assumed to be happening in public, so every eavesdropper knows the numbers

$$p,\ q,\ q^A \bmod p,\ q^B \bmod p.$$

However, similar to how factoring large integers is a hard problem, determining A (or B) given the numbers p, q and $q^A \bmod p$ (or $q^B \bmod p$) is an extremely hard mathematical problem. Since $\log_a(b)$ is the power that answers the question:

"to what power must I raise a to get b?"

the problem of determining A given p, q and $q^A \bmod p$ is a lot like taking a logarithm. It is therefore called the discrete logarithm problem, and is famously difficult to solve.

One problem with this streamlined method:

If Alice tries to send a pair (p, q) to Bob to start the key exchange, this communication can easily be intercepted by an eavesdropper Eve, who could pretend to be Alice and relay incorrect information to Bob. Unlike RSA (which allowed for digital signatures using the public/private keys of Alice and Bob), there is not any way of securing the key exchange against an eavesdropper like Eve. The D-H key exchange therefore has to be used alongside some additional digital signature method in order to guard against these attacks.

Example: Alice chooses $p = 577$, $q = 49$, $A = 123$ and sends

$$p = 577,\ q = 49,\ 49^{123} = 99 \bmod 577$$

to Bob. Bob chooses $B = 421$ and sends

$$49^{421} = 254 \bmod 577$$

back to Alice. In private, Alice computes:

$$254^{A} = 254^{123} = 46 \bmod 577$$

and in private, Bob computes

$$99^{B} = 99^{421} = 46 \bmod 577.$$

In this way they both know the key of $k=46$.
The public knows the numbers

577, 49, 99, and 254.

They cannot deduce 46 from these without solving the discrete logarithm problem.

This whole time we have neglected a discussion of why RSA actually works. That is, when Alice sends $m^{e_B} \bmod n_B$ to Bob

(e_B in the exponent, n_B after mod; (e_B, n_B) Bob's public key)

why is it that $(m^{e_B})^{d_B} = m \bmod n_B$ (here, (d_B, n_B) is Bob's private key). Specifically, we want to prove:

RSA theorem: Suppose p, q are primes and $n = pq$ is their product. If e, d are integers such that $ed = 1 \bmod \varphi(n) = 1 \bmod ((p-1)(q-1))$, then

$$(m^e)^d = m \bmod n$$

for every integer m with $0 < m < n$.

Proof of why RSA works:

Our proof will use Fermat's little theorem, namely that $\gcd(a,n)=1$ implies $a^{n-1} = 1 \bmod n$ (which, if you recall, we proved!).

We start with e, d that satisfy

$$ed = 1 \bmod (p-1)(q-1),$$ which means that

$$ed = 1 + k(p-1)(q-1) \quad \text{for some integer } k.$$

Now we want to show that $m^{ed} = m \bmod pq$, and from the above equation this means we want

$$m^{1+k(p-1)(q-1)} = m \cdot \left(m^{(p-1)(q-1)}\right)^j = m \bmod pq.$$

If it turns out that $m^{(p-1)(q-1)} = 1 \bmod pq$, then we'll have succeeded, because the equation above becomes:

$$m \cdot (1)^j = m \bmod pq.$$

We apply Fermat's little theorem twice to reach this conclusion.

First we write

$$m^{(p-1)(q-1)} = \left(m^{(p-1)}\right)^{q-1}$$

so Fermat's little theorem gives

$$m^{(p-1)(q-1)} = (1)^{q-1} = 1 \bmod p.$$

In the same way, Fermat's little theorem gives

$$m^{(p-1)(q-1)} = (1)^{p-1} = 1 \bmod q.$$

This means that

$$m^{(p-1)(q-1)} = 1 + kp$$

and $m^{(p-1)(q-1)} = 1 + lq$

for some integers k and l. But this means

$$kp = m^{(p-1)(q-1)} - 1 = lq.$$

But if $m^{(p-1)(q-1)} - 1$ is a multiple of both p and q, then it's a multiple of pq. So there's an integer j such that

$$m^{(p-1)(q-1)} - 1 = jpq.$$

But this means $m^{(p-1)(q-1)} = 1 + jpq = 1 \bmod pq$.

This is exactly what we were aiming to show, so the RSA theorem holds.