Public Key Cryptography

Public Key cryptography: The basic idea is to do away with the necessity of a secure key exchange, which is necessary for all private key encryption schemes. The idea is as follows:

1) Bob creates two keys, a public key, E and a private key D.

2) Bob posts the public key in a location that anyone can access.
The important thing here is that the knowledge of E does not compromise the value of D in any way shape or form.

3) Now, anyone who wants to send a message to Bob encrypts it using the public key E.

4) Bob can now read the message using his private key D. However, since the value of E gives no information as to the value to D, all others can not read the message.

The idea seems easy enough, but the difficulty is in finding some mathematical function to use in this scheme such that the knowledge of E does not compromise the secrecy of D. Clearly in all the other schemes we have seen, knowledge of the encrypting key all but completely gives away the decrypting key.

One thing to note however is that if you use a system outlined above with nothing extra, although Bob can decipher a message sent to him, he can not be sure of who the sender is, because the whole public has the ability to encipher a message, so someone could easily indicate in their message that they were someone else and Bob would not have any way of knowing. But, the person sending the message can be confident that no one read can read the plaintext except for Bob, the only person with the private key.

RSA Cryptosystem

Now, we are ready to look at the RSA algorithm:

1) Pick large primes p and q.
2) Compute $n=pq$ and $\phi(n)=(p-1)(q-1)$
3) Pick a value e such that $\gcd(e, \phi(n)) = 1$. (Note that this is fairly easy to do by randomly picking values e and testing them with Euclid's algorithm until you find one that works.)
4) Compute d such that $ed \equiv 1 \pmod{\phi(n)}$. You are guaranteed to be able to do this by the extended Euclidean algorithm.
5) Public keys : e, n
Private key : d, (n is also necessary for decryption, but is clearly public...)

Encryption function : $E_{n,e}(x) = x^e \pmod n$
Decryption function : $D_{n,d}(y) = y^d \pmod n$

Now, we must verify that this is a valid encryption scheme:

$D_{n,d}(E_{n,e}(x)) = D_{n,d}(x^e) = (x^e)^d \pmod n = x^{ed} \pmod n$

Now we will invoke the given information about the product ed:

$ed \equiv 1 \pmod{\phi(n)}$, thus, we can find an integer k such that $ed = k\phi(n) + 1$.

Now we have the following:

$$\begin{aligned} x^{ed} \pmod n &= x^{k\phi(n)+1} \pmod n \\ &= x^{k\phi(n)}x^1 \pmod n \\ &= (x^{\phi(n)})^k x \pmod n \\ &= (1)^k x \pmod n, \text{ invoking Euler's formula.} \\ &= x \pmod n, \text{ proving that the encryption scheme is valid, assuming that} \\ &\qquad \text{the } \gcd(x,n) = 1. \end{aligned}$$

Since you are picking large primes the probability that gcd(x,n) =1 is quite high. (If you pick 20 digit primes for both p and q, the probability is roughly $1 - 10^{-20}$ that gcd(x,n) = 1.

Diffie-Hellman Key Exchange

Motivation for Key Exchange Algorithm

The fastest encryption schemes are symmetric encryption schemes. In order to use these, the two communicators (Alice & Bob) would have to meet in a secure location. BUT, this sort of defeats the purpose, because their whole goal is to communicate when securely when they aren't in the same place. A modern day example of why it would be useful to exchange a key without meeting deals with an online purchase. You are making the purchase online so that you DON'T have to go to the store to meet with the vendor. So, you don't really want to go to the store to just exchange a secret key either. You want to be able to do that from the comfort of your own home.

Thus, that is the underlying problem: how do two users exchange a secret key without a secure communication channel?

The first solution to this problem was the Diffie-Hellman Key Exchange. What's interesting about this algorithm is that neither user actually gets to choose the key. But, at the end of the algorithm, both users have calculated the same key, which is not easy for an eavesdropper to calculate.

In order to understand why the Diffie-Hellman Key Exchange is difficult, it is important to understand the Discrete Log Problem.

Discrete Log Problem

The (discrete) exponentiation problem is as follows: Given a base a, an exponent b and a modulus p, calculate c such that $a^b \equiv c \pmod{p}$ and $0 \leq c < p$.

It turns out that this problem is fairly easy and can be calculated "quickly" using fast-exponentiation.

The discrete log problem is the inverse problem:

Given a base a, a result c ($0 \leq c < p$) and a modulus p, calculate the exponent b such that

$a^b \equiv c \pmod{p}$.

It turns out that no one has found a quick way to solve this problem. To get an intuition as to why this is the case, try picking different values of a and p, and listing out each successive power of a mod p. What you will find is that there is no discernable pattern for the list of numbers created. Thus, given a number on the list, it's very difficult to predict where it appears on the list.

Here's a concrete example:

Given a = 2, b = 7 and p = 37, calculate c: $2^7 = 128$, and $128 \equiv 17 \pmod{37}$

Given a = 2, c = 17, and p = 37, calculate b: Try each value of b until you find one that works! (For large prime numbers, this is too slow.)

Diffie Hellman Key Exchange

Alice and Bob agree on two values: a large prime number p, and a generator g, $1 < g < p$. (It's better if g is an actual generator, meaning that when you raise it to the 1st, 2nd, 3rd, …, p-1 powers, you get all different answers.)

These values are known to everyone.

In secret, Alice picks a value a, with $1 < a < p$.
In secret, Bob picks a value b, with $1 < b < p$.

Alice calculates $g^a \pmod p$, call this f(a) and sends it to Bob.
Bob calculates $g^b \pmod p$, call this f(b) and sends it to Alice.

Note that f(a) and f(b) are also known by everyone.

In secret, Alice computes $f(b)^a \pmod p$ – this is the exchanged key.
In secret, Bob computes $f(a)^b \pmod p$ – this is again, the exchanged key.

Why does this work?

$f(b)^a \equiv (g^b)^a \equiv g^{ab} \pmod p$. Similarly,
$f(a)^b \equiv (g^a)^b \equiv g^{ab} \pmod p$.

Here's a concrete example:

Let p = 37 and g = 13.

Let Alice pick a = 10. Alice calculates $13^{10} \pmod{37}$ which is 4 and sends that to Bob.

Let Bob pick b = 7. Bob calculates $13^7 \pmod{27}$ which is 32 and sends that to Alice.

(Note: 6 and 7 are secret to Alice and Bob, respectively, but both 4 and 32 are known by all.)

Alice receives 32 and calculates $32^{10} \pmod{37}$ which is 30, the secret key.
Bob receives 4 and calculates $4^7 \pmod{37}$ which is 30, the same secret key.

Note that neither Alice nor Bob chose 30, but that they ended up with that secret key anyway. Furthermore, note that even with knowing p = 37, g = 13, f(a) = 4 and f(b) = 32, it is difficult to ascertain the secret key, 30 without doing a brute force check.

In particular, if the discrete log problem were easy, this scheme could be broken. Consider the following:

If an adversary saw that f(a) = 4, p = 37 and g = 13 and could solve the discrete log problem, then they could calculate that $13^{10} \equiv 4 \pmod{37}$. Once they had this value, 10, then they could take f(b) = 32 and then calculate $32^{10} \pmod{37}$ to arrive at 30.

Thus, the Diffie-Hellman Key Exchange is only as secure as the Discrete Log problem.