CptS 464/564
Lecture 27

6th December 2004

Public-key cryptography

What is public-key cryptography?

Based on the observation that maybe there exist functions for which $F(x)$ is easy to compute but $F^{-1}(x)$ is very hard to compute (*i.e.* would take thousands or millions of years). These are called *one-way* functions. For cryptography we also need there to be a secret, *trap door*, that makes computing F^{-1} easy. Example: the corner post box: *anyone* can put mail in, but only the postal worker with a key can take mail out easily.

The RSA trap-door function

R: Rivest, S: Shamir, A: Adelman. This was the second trap-door function proposed for public-key cryptography (The first, called the Merkle-Hellman scheme and based on the well-known knapsack problem was broken within a few years of its proposal. But Merkle-Hellman was the inspiration for others to pursue the whole idea of public-key cryptography.)

The trap-door property in RSA comes from the difficulty of factoring large numbers: it is easy to multiply two large primes to get a large composite number, but it is difficult to recover the prime numbers given only the composite number.

Choose two large random prime numbers, p and q and let $n = pq$. Choose e so that e and $(p-1)(q-1)$ are relatively prime. The pair (n,e) is the encryption key. The create the cipher text c for message m compute $c = m^e (\text{mod } n)$. For decryption we use $d = e^{-1} (\text{mod } (p-1)(q-1)$ which is computed using Euclid's algorithm. Decryption takes place by computing $m = c^d (\text{mod } n)$, so the private key is (c,n).

Example: (from Schneier *Applied Cryptography)*

$$n = (47)(71) = 3337$$

e must be relatively prime to (46)(70)=3220.Let's try $e = 79$. Calculate

$$d = 79^{-1} (mod\ 3220) = 1019$$

Publish $(n,e) = (3337,79)$. Destroy p and q because disclosure of them would break the cipher.

To send me a message, a correspondent uses my *public*key to encipher the message. Let's try the message 688.

$$c = 688^{79}\ mod\ 3337 = 1570$$

$$m = 1570^{1019} mod\ 35 = 688$$

Notice that the algorithm is symmetric in d and e. This means that we could equally well use the private key d for encryption and the public key, e, for decryption. This leads to the notion of *signing:* a message that is successfully decrypted using e must have been encrypted by the secret d which by its secrecy is associated with a single individual.

Authentication using PK cryptography

Last time the question came up of "how are shared secret keys delivered from one party to another?" and we talked about out-of-band delivery methods such as couriers. Public-key methods alleviate some, but not all of the pain. Because public keys are public there is no need for a confidential channel to deliver them; but

there is still need for the channel to be authenticated: encryption using a public-key system does me no good if in fact I am encrypting messages with the public key of my enemy thinking it is that of my friend.

Like in symmetric cryptography a KDC is still used, but in a different way. Now the KDC may be called a Certificate Authority (CA) which signs a message saying "public key K_c belongs to real-world person or company c"by encrypting it using the CA's private key.The CA should only sign a public key upon proof that it truly belongs to the claimed real-world entity. (There have been instances of major CAs signing false certificates. Apparently it wasn't at the time very hard to get them to do it.)

Anyone who receives the certificate can confirm that the CA has made this statement about K_c . Who vouches for the CA's signature? Widespread publication of its public key, *e.g.* as part of the distribution of a particular program or another CA or by communication outside the system.

If c and v both have CA-signed certificates for their public keys and they can authenticate each other and establish a session key (simple protocol, there are better – see p. 460 *ff*):

c sends to v: timestamp and session key encrypted in v's public key and signed with c's private key. c may send its own certificate to v as well.

Note that the session key is almost always a shared-secret key because the execution cost of public-key encryption algorithms (assymetric algorithms) is very significantly more than that of symmetric-key algorithms.

Advantages: CA need not be on-line when c and v establish their session; CA must be secure, but really only has to protect one secret: it's own private key.

Another cool trick: secret sharing

Suppose you have a very valuable secret for which either loss or disclosure would be disastrous. Entrusting such a secret to a single copy or person is very risky, yet making multiple copies of the secret increases the risk of disclosure. A simple mathematical fact provides the basis for a solution: a polynomial of degree $n-1$ is determined by n points. This allows the secret to be broken up into m pieces so that any n of the pieces can be used to reconstruct the secret. To share secret s choose a random polynomial

$$f(x) = a_1x^1 + a_2x^2 + \ldots + a_{n-1}x^{n-1} + s$$

Construct pairs $(1, f(1))$, $(2, f(2))$, ..., $(m, f(m))$. Now, any n of these pairs can be used to reconstruct s by solving the set of simultaneous linear equations (Lagrangian interpolation):

$$\begin{bmatrix} i_1^1 & \dots & i_1^{n-1} \\ \dots & \dots & \dots \\ i_n^1 & \dots & i_n^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ \dots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} f(i_1) \\ \dots \\ f(i_n) \end{bmatrix}$$

Once $a_1 \dots a_{n-1}$ are known it is a simple matter to recover s.

Example:

$$s = -22$$

$$f(x) = 14x + 31x^2 - 22$$

$f(1) = 23$, $f(2) = 130$, $f(-1) = -5$, $f(-2) = 74$. Pick any three of these and write

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & -2 & 4 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 23 \\ -5 \\ 74 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 0 \\ 0 & 3 & -3 \end{bmatrix} \begin{bmatrix} s \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 23 \\ 28 \\ -51 \end{bmatrix}$$

So $a_1 = 14$, $3(14) - 3a_2 = -51$, so $a_2 = -93/-3 = 31$ and $s + 14 + 31 = 23$. So $s = -22$.

Use of PK techniques in distributed systems

As a system builder you certainly should not be inventing your own cryptographic methods: this is a field for experts (but it's fine to want to become an expert.) Any system built on keeping its *methods* secret (security by obscurity) is now generally considered doomed. Quality cryptography involves published and publically tested *algorithms* for which only the *keys themselves* need be kept secret.

It's also worth remembering that "those who think that their problem can be solved by simply applying cryptography don't understand cryptography and don't understand their problem" (Roger Needham or Butler Lampson; each credits the other).

Nowadays, cryptographic modules are pretty freely available: Java has an extensive security infrastructure, similar things exist for Perl and Python. For C programming the OpenSSL library has very useful support for public-key based authentication including checking of certificates, as well as support for various session-encryption schemes using symmetric-key encryption. So there is little justification any more for even writing your own *implementation* of encryption.