Chapter 1

Public-key cryptography

1.1 Going public

1.1.1 Symmetric crypto vs public-key crypto

Symmetric cryptography is the oldest kind of cryptography. Indeed, the idea of having a public key is quite disturbing at first glance. Just a quick reminder, here is how we settle a supposed secure way of communication between Ali and Bachar using public-key crypto.

1. Ali and Bachar both generate a key-pair : they keep one private for each, and they publish the other, making it public, so anyone has access to Ali and Bachar's public key.
2. If Bachar wants to send a message to Ali, he **encrypts with Ali's public key** and sends him
3. Ali receives messages that are encrypted using his public key. He decrypts them using his private key.

That's the kind of communication we have for public-key crypto. Symmetric crypto was based on the idea of having **only one key**, kept secret, between A and B.

1.1.2 Why going public ?

There are many problems to symmetric crypto :

- Key distribution problem : the key must be established between Ali and Bachar. How do they do ? The communication link between them is not secure.
- Number of keys : if there are n users, there are $n(n-1)/2$ keys. That's a lot ! 4 million keys for 2000 people.
- No protection against cheating : as Ali has the same secret key as Bachar, he can create a message from scratch and claim that Bachar created it.

In addition to all this, let me talk a little bit, because you guys talk too much,  . Symmetric crypto algorithms are meant to avoid any compact mathematical description. That is easy shit. Asymmetric crypto is **built on one-way number-theoric functions so there is no bullshitting in their security**, you get me ?

1.1.3 Actually going public

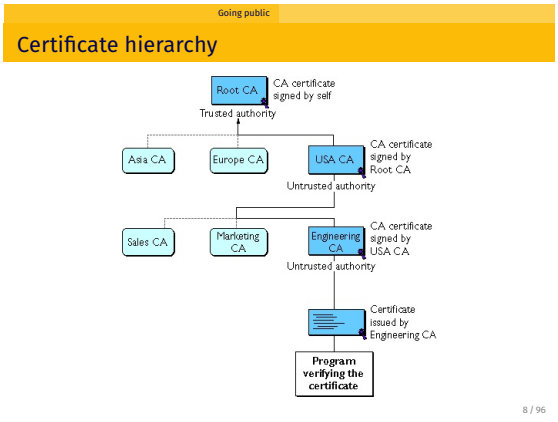
When going public, we are immediately subject to a particular kind of attack : the **Man-In-the-Middle attack** (MIM[1]). In public-key crypto, a crucial part is the exchange of keys : Ali gives his public key to Bachar, so that Bachar can send him messages, and inversely. But there can be an outsider Omar that acts as follows :

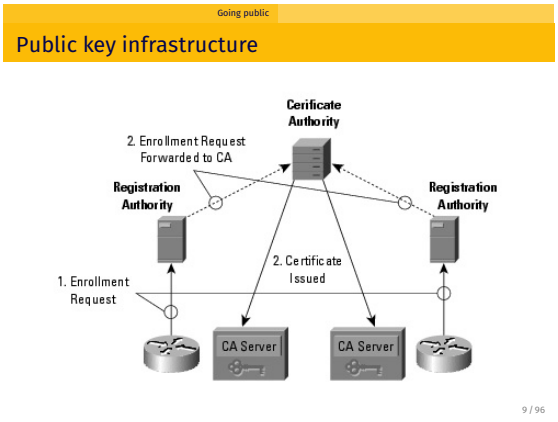
- He captures Ali's public key k_A before Bachar gets it
- He sends to Bachar a false public key, being his own, k_O
- Bachar sends his public key to Ali but it gets intercepted by Omar, that gives to Ali his k_O instead.
- Ali thinks he is receiving messages from Bachar, and inversely, but it is actually the no-life Omar that is intercepting everything and manipulating the conversation.

To avoid this, we make use of **certificates**, objects that bind a public key to an identity. A public key certificate is an object that contains :

- The public key
- Information allowing to identify its owner, such as the name, the ID, IP, etc.
- A digital signature on the key and the information : this is signed by a **certification authority**.

Thanks to this, at the end, Omar can not generate a fake certificate using his key and Ali's information because the signature will be different than Ali's public key certificate, and Bachar will notice that. The certification authority must of course be trustworthy. We can also build hierarchies between CAs as seen below.





Let us recall that we solve only a part of the problem, because Omar can still intercept the communication between Ali and the CA ! Hence, we must have a **secure authenticated channel with the CA**.

[1]Pas MYM, mécréant.

1.1.4 Hybric encryption

Note that in practice, we use **hybrid encryption** to improve both efficiency and bandwith (avoid transmitting a lot of data).

Going public

In practice: hybrid encryption

To improve both efficiency and bandwidth we can combine asymmetric encryption with symmetric encryption.

Alice computes the encryption c of the message m intended for Bob in the following way:

1. Alice chooses randomly the symmetric key k;
2. Alice computes $(c_k, c_m) = \left(\mathsf{Enc}_{\text{Bob's public key}}(k), \mathsf{Enc}_k(m)\right)$.

Bob decrypts as follows:

1. Bob recovers $k = \mathsf{Dec}_{\text{Bob's private key}}(c_k)$
2. Bob recovers $m = \mathsf{Dec}_k(c_m)$

12 / 96

There are many ways of doing public-key crypto, based on different mathematical problems. There is **large integer factorization** ($\rightarrow$ RSA), **discrete logarithm problem** ($\rightarrow$ ElGamal DSA, and DH), and **elliptic curves** ($\rightarrow$ ECDSA)

1.2 RSA : a factorization problem

RSA is mainly about key generation. It is an interessant way of generating keys but is subject to a lot of attacks.

1.2.1 Key generation

The key generation relies on the fact that factorizing large integers is complicated. Hence, it is based on two prime numbers p and q, kept private but **their product n is kept public**, as part of the public key.

The other part of the public key is an exponent that will come in play when encrypting. We note it e. It is public, and satisfies

$$3 \leq e \leq (p-1)(q-1) - 3 \qquad \gcd(e, (p-1)(q-1)) = 1\,.$$

Often, one chooses e and then generates the primes.

The private key is the inverse of e modulo $(p-1)(q-1)$.

Rivest-Shamir-Adleman

RSA key generation

The user generates a public-private key pair as follows:

- Privately generate two large distinct primes p and q
- Choose a public exponent $3 \leq e \leq (p-1)(q-1)-3$
  - it must satisfy $\gcd(e,(p-1)(q-1)) = 1$
  - often, one chooses $e \in \{3, 17, 2^{16}+1\}$ then generates the primes
- Compute the private exponent $d = e^{-1} \bmod (p-1)(q-1)$
- Compute the public modulus $n = pq$ (and discard p and q)

The public key is (n, e) and the private key is (n, d).

26 / 96

1.2.2 RSA textbook encryption

Here is an encryption scheme that stinks a lot : we elevate the message to the power of e to get the ciphertext. To decrypt the ciphertext, we elevate it to the power of d, as e is the inverse of d.

1.2.3 RSA textbook signature

It is basically similar, but here we are **signing** a message with the private key, so $s = m^d \bmod n$. Again, to check the signature, we can elevate s to the power of e to verify if we indeed get the message back (valid signature) or not.

Rivest-Shamir-Adleman

RSA "textbook encryption" (don't use!)

From plaintext $m \in \mathbb{Z}_n$ to ciphertext $c \in \mathbb{Z}_n$ and back:

- **Encryption:** $c = m^e \bmod n$
- **Decryption:** $m = c^d \bmod n$

Why is it correct?

$$\begin{aligned} c^d &\equiv (m^e)^d \equiv m^{ed} \\ &\equiv m^{ed \bmod \Phi(n)} && \text{by Euler's theorem} \\ &\equiv m^{ed \bmod (p-1)(q-1)} && \text{since } p \text{ and } q \text{ are distinct primes} \\ &\equiv m^1 && \text{by definition of } e \text{ and } d \\ &\equiv m \pmod{n} \end{aligned}$$

27 / 96

Rivest-Shamir-Adleman

RSA "textbook signature" (don't use!)

From message $m \in \mathbb{Z}_n$ to signature $s \in \mathbb{Z}_n$ and back:

- **Signature:** Send (m, s), with $s = m^d \bmod n$
- **Verification:** Check $m \stackrel{?}{=} s^e \bmod n$

28 / 96

1.2.4 Attacks on RSA

RSA is subject to many attacks, one of the easiest ones being the **short message attack**, when we know that e is small. We can send short messages so the modular reduction by n does nothing in the encryption. So we can find back m by taking the e-th root of $c = m^e$. One just has to try the different cs.

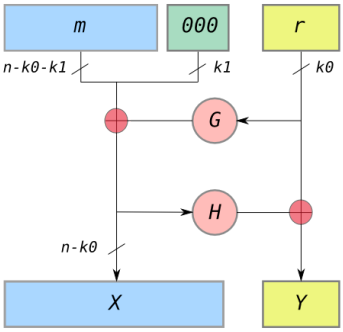
1.2.5 Powering RSA

RSA-OAEP : optimal asymmetric encryption padding

Padding is really interesting. We here add padding to the original message, and randomness to the pre-processing of the message before encryption. Encryption is done as always, exponentiating by e, moduling by n, but the input is a new one : the message after pre-processing.

Rivest-Shamir-Adleman | RSA for confidentiality

RSA-OAEP: Optimal asymmetric encryption padding



[Mihir Bellare and Phillip Rogaway, Eurocrypt 1994]

Encryption: m ... then $c = (X\|Y)^e \bmod n$
Decryption: $(X\|Y) = c^d \bmod n$, then ... m

RSA-OAEP

- makes the scheme probabilistic;
- prevents that an adversary recovers any portion of the plaintext without being able to invert RSA;
- is shown to be IND-CCA secure, assuming that G and H behave as random oracles.

32 / 96

This helps with the short message attack, makes the scheme probabilistic, and introduces diffusion. In addition, it is shown to be IND-CCA secure. Wow !