Public Key Infrastructure (PKI)

COMPUTER NETWORKS A.A. 24/25

Leonardo Maccari, DAIS: Ca' Foscari University of Venice,
leonardo.maccari@unive.it

Venice, fall 2024

Sect. 1 Distributing Public Keys

Distributing Public Keys

First of all, what is a public key:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
mQINBFJ03JYBEADtKuOxyMibiwnKZw6rD7gjmRi4UTaIA3v9ro0gUU0r51g7Eu1p
DWjUjZ4ReXkZyQ20YXK0ozV5CxMEsiOcN92tgAhwK1fgqZrz2/N4NJdbzY/Y56zI

[...]

C6oR0TqUsO3sZRj6ITJyaZGrY1y/RrzlH9xtLBUqyVTmeAx+B5GWQgvoaPFRsbMy
/r3xp+vOU48NRu0dSt/TowN1p1QDdIfeDYBrbeGbFzY+MiFaX0DrhHDwRA==
=pilJ
-----END PGP PUBLIC KEY BLOCK-----
```

. . . a bunch of apparently random binary data.

A Key is a File

- Essentially it is just a file containing a seemingly random string
- When generating a key it is impossible to decide its content beforehand, so the key can't contain useful information.
- For this reason besides the key itself, the file contains some meta-data: name, e-mail address and other identifying information.
- Still this does not guarantee that the key was generated by those that claim to have generated it.
- There are several ways to associate a key to an identity.

Informal Method 1: Key Fingerprint

1. Once Alice generates her keys, she publishes the hash of the public key anywhere on the Internet: her website, the signature of her email, her card, linkedin account... This is called a key fingerprint.
2. When Bob receives the key, he controls that the fingerprint is the same one that Alice sponsored in mailing lists, personal page etc... eventually before they got in contact.
3. This technique does not scale well enough to be used on the web, but in informal groups it is OK

My Card


VENETIARVM VNIVERSITAS
IN DOMO FOSCARI
Ca' Foscari
University
of Venice
Leonardo Maccari
Associate Professor
Department of Environmental
Science, Informatics and Statistics
via Torino, 155
30172 Venice, Italy
Tel: +39 041 234 8476
leonardo.maccari@unive.it
http://www.dais.unive.it/~maccari/
GPG: 567C B5B1 65A3 5295 ED83
1732 B425 3D8A AABE 2BD7

My Card


VENETIARVM VNIVERSITAS
IN DOMO FOSCARI
Ca' Foscari
University
of Venice
Leonardo Maccari
Associate Professor
Department of Environmental
Science, Informatics and Statistics
via Torino, 155
30172 Venice, Italy
Tel: +39 041 234 8476
leonardo.maccari@unive.it
GPG: 567C B5B1 65A3 5295 ED83
1732 B425 3D8A AABE 2BD7

Informal Methods 2: Key Servers

- On the Internet you can find keyservers on which to upload your public key, together with meta-data.
- Public keyserver use a synchronization protocol, so what you upload to one is mirrored on others.
- The keyserver does not guarantee anything, it is not responsible for establishing the association between user and key.
- It simply accepts the upload and download of the keys themselves.
- The keyserver also accepts information such as the user name, address, e-mail, so by searching in the keyserver search engine you can search the key you need.

Informal Methods: Key Servers

- Note Well: finding the key of Joe Biden on a keyserver does not mean having the certainty that the American president really uses that key.
- The keyservers are just a convenient way to keep public keys, so that when Bob have Alice's key fingerprint he knows where to look for it.
- Alice may have uploaded it to a key server, which makes it easier for Bob to find it, instead of googling for it and getting it from a random source.
- A keyserver: `https://keyserver.ubuntu.com/`

Live Session

- `gpg -gen-key`
- `gpg -list-key KEY_ID`
- `gpg -send-key KEY_ID`
- GPG creates a primary key that is used for digital signatures and a secondary one to encrypt.
- This is fundamental for the security of the system, the first one is used rarely and and you should keep it in a safe computer, the second one is used frequently
- We will describe a similar organization for a real Certification Authority: Let's Encrypt

A Note on Random Numbers

- The modulo n of RSA should never be reused.
- A trivial reason is: if Eve generates a key with a certain n, and later on Alice generates another key with a certain n, even if e might be chosen differently, Eve can factor n and invert e.
- There is a problem even if Alice uses twice the same n with different e[1]
- So n should be chosen at random. How is this achieved? loop on:
  - Pick a random big integer p
  - Make a primality test
- The density of prime numbers is high enough, and primality tests are polynomial (checking the primality is simpler than factoring). So this is feasible.

[1] See this nice stackexchange explanation.

Informal Methods 3: Web of Trust

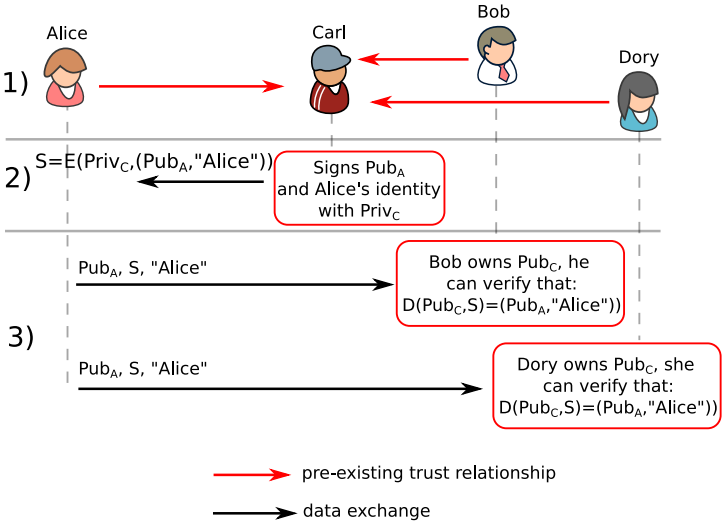
- A Web of Trust (WoT) is a social network of contacts through which the participants certify the identity of others.
- The key point is that Bob and Alice trust Carl. This means essentially two things,
  - They own its public key
  - They trust he will not misbehave, so if he certifies something, Bob and Alice believe it is true.
- Carl has a trust relationship with both Alice and Bob, but Alice and Bob do not trust other, Carl can certify to Bob that `Pub_A` belongs to Alice.
- Given this, how does certification technically happen?

Informal Methods 3: Web of Trust

1. Preliminaries: Alice and Carl meet and exchange their fingerprints, the same happens between Bob and Carl.
2. Carl provides to Alice a key signature:

$$S = E(\texttt{Priv\_C}, (\texttt{Pub\_A}, \text{“}Alice\text{”}))$$

   You read this as: *Carl encrypts both `Pub_A` and the identity of Alice (her name) with `Priv_C`.*
   Everybody owning `Pub_C` can decrypt S. If they do they conclude that: *Carl certifies that `Pub_A` belongs to a person named Alice*. If they trust Carl, they now trust that `Pub_A` belongs to Alice.
3. Alice sends `Pub_A` and S to Bob (or Bob downloads them from a key server), Bob uses `Pub_C` to verify S. Since Bob trusts Carl, he now trusts that `Pub_A` belongs to Alice.
4. Also Dory trusts Carl, she can do the same.


Alice
Carl
Bob
Dory
1)
2)
S=E(Priv$_C$,(Pub$_A$,"Alice"))
Signs Pub$_A$ and Alice's identity with Priv$_C$
3)
Pub$_A$, S, "Alice"
Bob owns Pub$_C$, he can verify that: D(Pub$_C$,S)=(Pub$_A$,"Alice"))
Pub$_A$, S, "Alice"
Dory owns Pub$_C$, she can verify that: D(Pub$_C$,S)=(Pub$_A$,"Alice"))
pre-existing trust relationship
data exchange

Important Details:

- Initially we need a trust relationship. This has two components:
  - Alice, Bob and Dory trust Carl. They know that he doesn't misbehave
  - They own the public key of Carl.
- After this is in place, Alice has a *signed* key, that is, she can send her key to someone together with S. **Carl is not required anymore**, he does not have to participate to the next exchanges of data.
- In fact, Alice can use her signed key with Bob and Dory, and anyone else that trusts Carl.

Informal Methods: Web of Trust

Transitively, trust can be moved over longer paths

- Since Dory trusts Carl, she can also sign his public key, and give the signature S_2 to Carl.

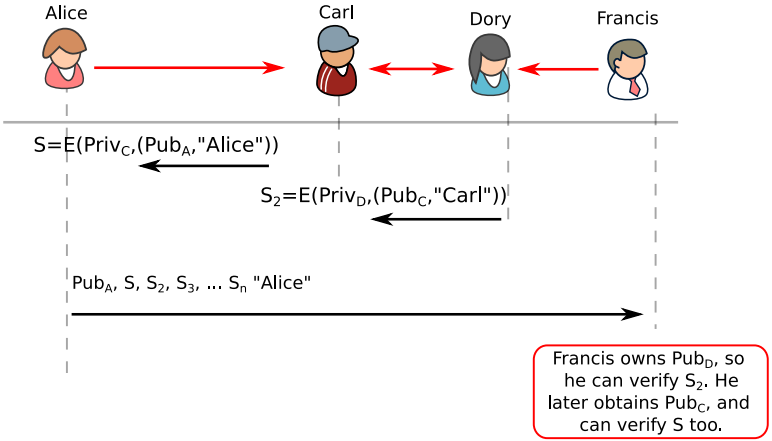
$$S_2 = E(\texttt{Priv\_D}, (\texttt{Pub\_C}, \text{“}Carl\text{”}))$$

- Francis trusts Dory, he does not trust Carl, and he has to communicate with Alice.
- Alice sends S to Francis, but he can not verify it, because he doesn't trust Carl.

Informal Methods: Web of Trust

- He then asks Alice: give me the list of people that trust Carl
- Alice provides the signatures she knows of Carl. In the list Dory appears, together with S_2
- Francis owns `Pub_D`, so he can verify S_2
- Now Francis has a chain of trust that certifies that `Pub_A` is Alice's key.

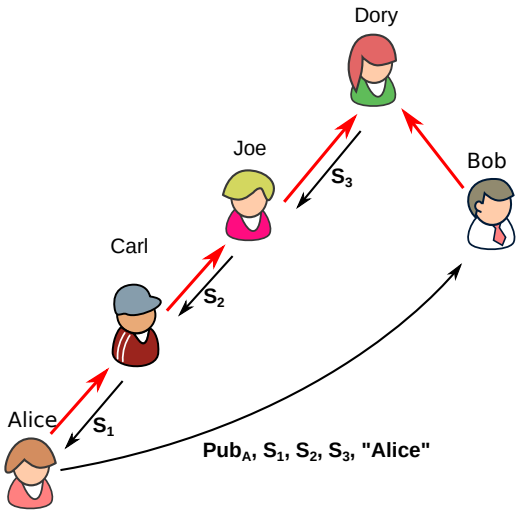
Informal Methods: Web of Trust



Some Notes

- The important concept here is not the way signatures and keys are actually moved around
- It does not matter the protocol used to ask/obtain them
- What matters is that a network graph is created among people
- As long as there is a path of trust that goes from Alice to Bob, they can communicate securely, because someone certifies the association between their keys and their identities

Trust Path



- Bob can verify S_3 because he owns Dory's key. And in cascade he can obtain the public key of Joe and verify S_2, then the same with Carl and verify S_1. Finally he is sure `Pub_A` belongs to Alice.
- Bob can Encrypt to Alice, not the other way around

Informal Methods: Web of Trust

A practical application

- Indeed, it is not really useful for people like you and me, it is way more useful for people with high visibility.
- Richard Stallman is well known open source developer.
- If you search for his GPG key you will find many signatures.
- Ideally, you could search for a path of trust from you to him and verify his key

WoT: Take Away

- WoT is used mostly in informal situations, but introduces two key concepts for the next solution:
  - Key signatures: the fact that an entity C can sign the public key of an entity A together with A's meta-data to certify that `Pub_A` belongs to A (A and B and C need not to be physical persons)
  - Trust delegation: if both A and B trust C than C can intermediate and be a certifier.
  - Trust is asymmetrical: you can trust someone, and thus you believe information that is signed with her private key, but she may not trust you, and she will not provide a signature for you.

Sect. 2 Public Key Infrastructure

Public Key Infrastructure

- A PKI is an infrastructure in which there is one party that is trusted by all the actors, a *trusted third party*.
- This entity has a public and private key, and everybody knows the public key of this entity.
- In the technical jargon this entity is called a Certification Authority (CA)

The CA Signs keys

- Both Alice and Bob trust the CA. This means that:
  - They trust the CA will not misbehave
  - They already know the public key of the CA
- Alice can ask a CA to sign `Pub_A`.
- When Bob receives Alice's public key, he checks that it is signed by the CA
- If this is true, then Bob trusts that `Pub_A` belongs to Alice.
- A signed key is called a **certificate**

Workflow for Obtaining a Certificate

1. Alice generates `Pub_A` and `Priv_A`.
2. Alice creates a Certificate Signing Request (CSR). A CSR is a file with a specific format that contains her information and `Pub_A`.
3. The CSR is signed with `Priv_A`.

$$\texttt{CSR} = \{\texttt{Pub\_A}, \text{“}Alice\text{”}, E(\texttt{Priv\_A}, (\texttt{Pub\_A}, \text{“}Alice\text{”}))\}$$

Workflow for Obtaining a Certificate (2)

4. Alice sends the CSR to the CA: for instance Poste Italiane is a well known CA
5. The CA checks the identity (Alice will have to show some document attesting her identity)
6. Since the CSR is signed with `Priv_A`, the CA uses `Pub_A` to verify the signature. Now the CA knows that Alice owns `Priv_A`
7. The CA returns a valid certificate: a certificate is the CSR, with the digital signature of the CA added.

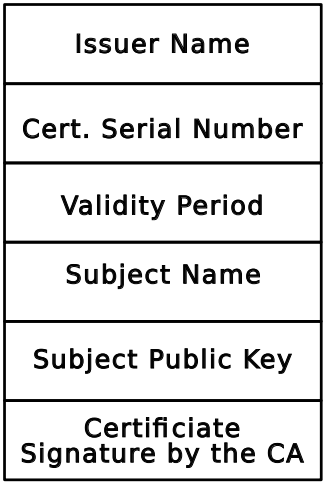
$$\texttt{Cert} = \{\texttt{CSR}, E(\texttt{Priv\_CA}, \texttt{CSR},)\}$$

8. Certificates use a standard format, called X.509

Certificate Example

Some of the important fields in a X.509 certificate:

- An identification number of the CA (unique)
- A serial number that identifies the certificate (unique)
- A validity time
- Data that identify the owner of the the key (name, IP address, domain...). Note that a certificate can be given to people, but also to Internet hosts and other entities.
- The public key
- The signature of the whole certificate by the CA, using the CA private key.



More Details

- This process is done only once per validity period.
- Once the certificate is delivered the CA is not needed anymore, it is not a bottleneck in the data communication
- The CA does not need to know the private key of the requester. This is very important, because the private key must be private.

Signing Details

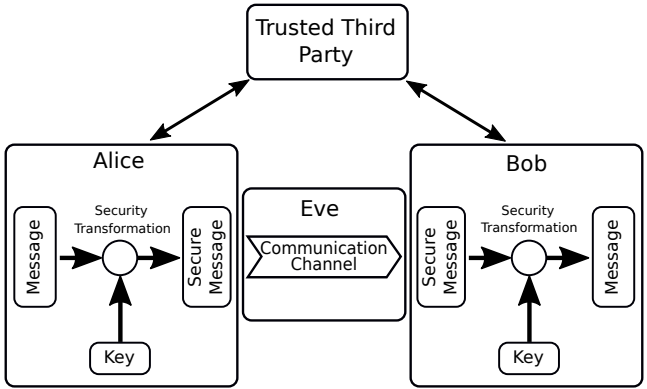
- We know that Public Key encryption is computationally heavy,
- The certificate contains `Pub_A` and many other information that can make it large.
- The signature of the CA is in reality the signature of a hash function applied to the certificate itself (obviously excluding the signature field).
- The certificate must also include the name of the function used to make the hash

Workflow for Receiving a Certificate

When Bob receives a valid certificate from Alice, signed by a CA, he does the following:

- Checks that the name reported in the certificate is Alice's name.
- Checks that the certificate is valid for the time being.
- Checks that the CA is a known CA. Bob has a database of valid CAs that he consider trusted, and for which he owns the public key.
- Decrypts the signature with the CA public key, and obtains a digest D of the certificate itself
- Computes an hash of the certificate D' (excluding the signature field)
- If $D = D'$ the certificate is valid and Bob now trusts Alice's public key.

Trusted Third Party



The CA is what we called the trusted third party in the attack model.

Certificates for Everything

- The concept of certificates can be applied to everything. Some examples are:
  - CAs provide certificates that are stored in an usb stick and can be used to prove your identity.
  - CAs can be used to make a network authentication. For instance when you log-in a Wi-Fi network that requests you to use the WPA-EAP/PEAP authentication, like Eduroam.
  - The most common use is to certify domain names.

Main Application: Domain Names

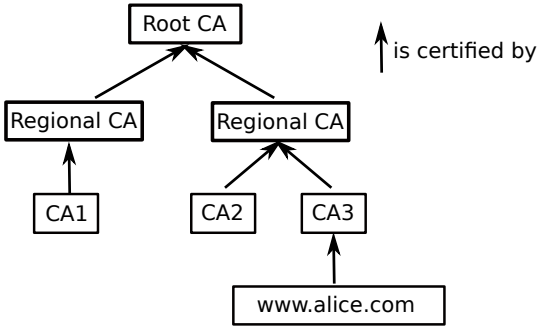
- In the certificate "owner" field, a domain name is contained: `www.alice.com`
- When Bob browses Alice's website using a secure communication, `www.alice.com` provides Bob `Pub_A`
- `Pub_A` is contained in a certificate. The certificate could be valid (Bob's browser shows a green lock) or invalid for some reason, and Bob receives a warning. Common reasons to be invalid could be:
  - Alice's certificate is released by a CA unknown to Bob's Browser
  - Alice's certificate is expired
  - Alice's certificate is for a different domain.
- In all cases, the browser will complain and try to prevent Bob from visiting the site.

What CAs does Bob Trust?

- Actually, there are many CAs in the world.
- There are legally valid CAs, like Poste Italiane, there are local CAs (you can set-up a CA for your own organization).
- How does Bob decide which CAs to trust?
- There are two ways: the first is that Bob has a list of trusted CAs.
- For instance, Bob's browser has a list of trusted CAs for which it knows the public key.

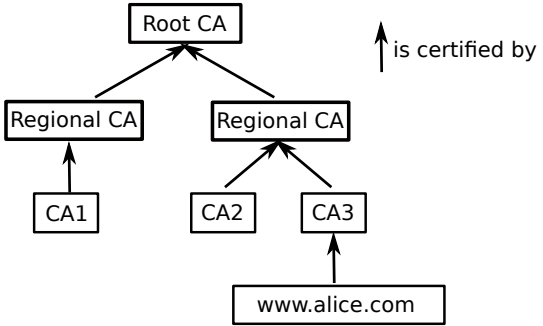
Certificate Chains

- The second is that Bob do not trust the CA, but the CA was itself certified by a CA he trusts.
- CAs can sign each others' certificates and create a **web of trust**, as we explained before.



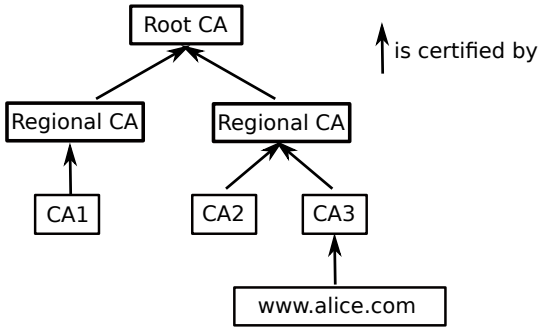
Certificate Chains

- The certificate authority at the highest level self-signs its certificate.
- A self-signed certificate is used only by a few "root CAs" and browsers to have direct trust on them, that is, they come with their certificates already installed.
- For instance, Microsoft, or Apple are root CAs and use certificates for their products.



Certificate Chains

- When Bob browses `www.alice.com` the webserver of Alice must provide not only its certificate
- It must provide all the certificates up to the root CA: 3 in this case.
- The browser of Bob receives all of them, he must own the self-signed root CA already, and checks the certificates to the leaf of the tree.



Not Mutual Trust

- This is a practical example of a-symmetric trust.
- At the end, the browser knows the public key of the web server, that uses that domain.
- The server does not know anything about the browser, so the communication is secured in one direction only, as we showed in the WoT example with Alice and Bob.
- Some other mean must be used to authenticate the user to the server, this is generally done with username and password.

Certificate Revocation Lists (CRLs)

Assume you own a CA. What happens in the following cases?

- one of your customers loses the laptop with a certificate and the private key
- one of your customer has an ex-employee that ran away with the private key
- one of your servers, that contains your private key, is compromised

In all these cases there is a need to revoke one or more than one certificate that you already issued.

CRLs

- CRLs are lists of certificates that have been revoked by the CA.
- A CRL is signed by the key of the CA itself, and published by the CA.
- Browsers should periodically update the CRLs from all the CAs they know.
- Yet there is no real agreement and unique standard protocol on the best way to do it.

PKI for Domain Names: Take away

- The chain of trust implemented by CAs is called a Public Key Infrastructure (PKI)
- It requires some root CAs to be known by every user, for instance, their public keys (in the format of a self-signed certificate) are stored in all browsers.
- The CA controls the identity of those that request a certificate, and releases a signed certificate.
- Those that receive a valid certificate can now safely associate a public key to an identity.

PKI for Domain Names: Take away

- Recall: when using public keys Alice and Bob need an authenticated channel to exchange public keys. Without that channel, they can be victim of MiTM attacks.
- Without a PKI Alice needs to use an authenticated channel to securely receive a public key from every website she browse, before she visits them. This is impractical.

The system of PKIs reduces the number of identities you have to trust: not every domain you browse, but just a small number of known CAs.

Do we trust CAs?

- In general, we have to.
- But CAs, as every other complex system can be insecure.
- When a CA is cracked, the consequences are enormous
- For instance, in 2011 DigiNotar was cracked, and fraudolent certificates were released.
- This means that someone connected to a website (i.e. google.com) and even if the browser was showing that there was no problem, the website was not the real one.
- The Dutch state was using certificates from DigiNotar, and many public services were simply interrupted.

CAs are complex

- Setting up a CA is complicate under a security point of View
- A nice description of the certificate chain inside a CA is provided by Let's Encrypt: `https://letsencrypt.org/certificates/`
- Let's review the organization of that CA. **This is part of the program**.
- Also Amazon AWS has instructions on how to set up a complex CA hierarchy

Exercise

What do we need to set-up a PKI and a client-server relationship?

- Go browse `www.unive.it`, click on the lock in the browser and answer the questions:
  - How long is the chain to the root CA?
  - Who is the root CA?
  - What is the length of the public key Unive uses?
  - What is the hash function it uses for the certificate?
- Now assume you are the system administrator of Unive.
  - List all the keys you need to set-up the web server of www.unive.it.
  - For every item in the previous question state who is the creator of it