Deliverable 3.3

Cyber Security Handbook for ICT Professionals

European Commission Tempus Project: 544088-TEMPUS-1-2013-1-SI-TEMPUS-JPHES

This project has been funded with support from the European Commission.

Table of content

1. ACCESS CONTROL FUNDAMENTALS

The expression *access control* denotes the collection of models, techniques, processes and policies that allow managers of a system to exercise a directing or restraining influence over the behaviour, use, and content of a system. In particular, all secure information systems require appropriate mechanisms in order to protect resources against unauthorized viewing, tampering or destruction. Access control permits management to specify what users can do, which resources they can access, and what operations they can perform on a system, and comprehends all security features that control how users and systems communicate and interact with one another.

Formally, the term *access* refers to any authorized flow of information. An *access mechanism* is therefore an instrument to establish if a flow of information is authorized, and *access control* is a set of access mechanisms established in order to protect a set of assets. Access control techniques are defined in terms of three fundamental concepts:

- Object: A passive entity that contains information
- Subject: An active entity that requests access to an object or the data in an object
- Access: The flow of information between subject and object

The continuous evolution of corporate environments, the difficult management of security tools, and the presence of low-skilled users are only a few of the obstacles to the design of a secure information system. Controlling access to data, especially in complex systems that include resources and users with different classifications, is of primary importance. The purpose of these notes is providing an overview of access control fundamentals, as well as a practical guide for both users and system managers to comply with the most advanced standards.

1.1 ACCESS CONTROL FUNCTIONS

Access control, whose flow is depicted in Figure 1.1, is based on the definition of four security functions:

- **Identification**: refers to the ability to determine who the subject is through the use of her user name or other public information
- **Authentication**: refers to the ability to determine whether the subject is actually who she claims to be, based on something a person is, has, or does (*e.g.*, through the use of biometrics, passwords, passphrase, token, or other private information)
- **Authorization**: refers to the ability to determine what a subject can do

- **Accountability**: refers to the ability to determine what a subject did

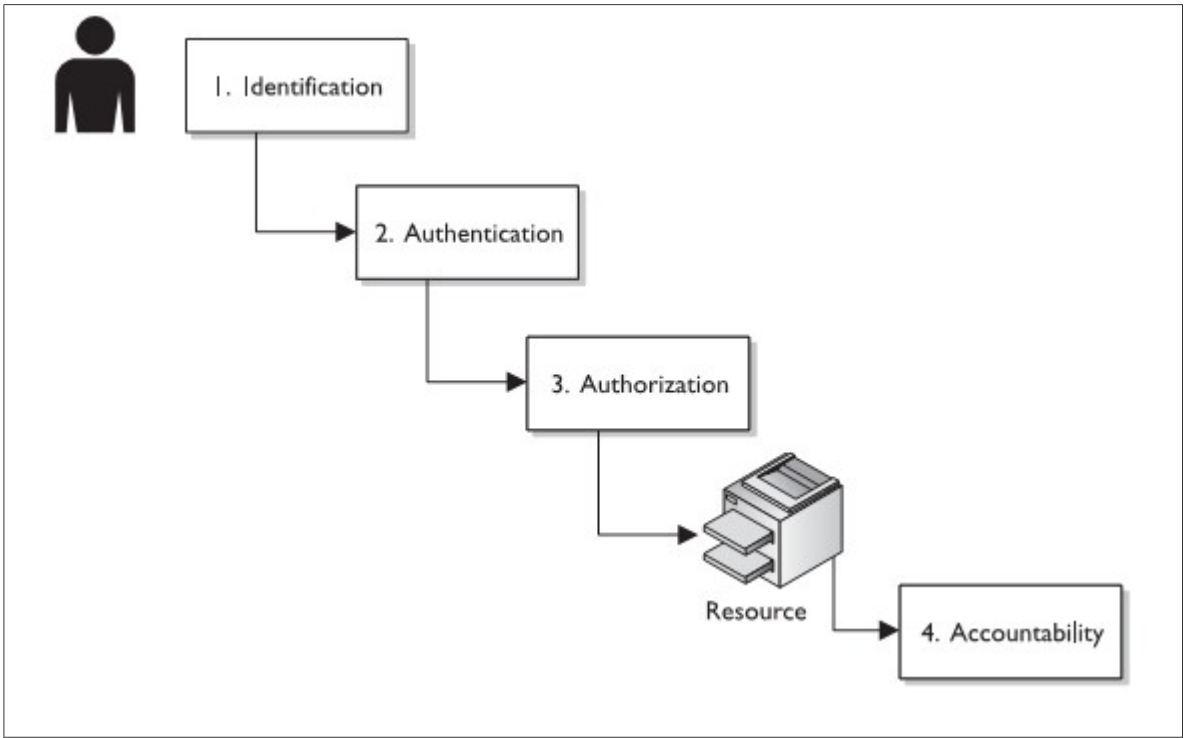

Figure 1.1 – A typical access control flow

Any method of establishing the subject's identity falls within the scope of identification, regardless of whether the subject is a person, a program or a process. Following four simple principles is usually sufficient for a secure identification system:

- Different users must have different usernames (uniqueness)
- Usernames or other identifiers must follow a standard naming scheme (standardization)
- The identifiers must not be descriptive of the user's position or tasks (anonimity)
- No identifier can be shared between users (secrecy)

After a user is identified, she needs to be authenticated, *i.e.*, she needs to *prove* that she indeed is who she claims to be. This is the most difficult step, and we will focus on authentication later.

Once the user has been authenticated, she must be given the correct authorizations. In most systems, there are three types of actions over data that a user can be authorized to do:

- *Read*: read file content and list directory content
- *Write*: add, create, delete, rename content of files and directories
- *Execute*: run a program

The main concepts to keep in mind when rights are granted to users are the following:

- *Privilege Creep*: a system's manager must be very careful to the gradual accumulation of access rights beyond what an individual needs to do his job
- *Default to Zero*: by default, any user is given no privileges
- *Need to Know* principle: access to the information is granted only if it is strictly necessary for the conduct of one's official duties

Finally, each user must always be accountable for what she did. To always know which user is responsible for an action, it is common practice to rely on *audit trails* (records) and *logs***:** proofs of an event (*e.g.*, security violations and incidents) that associate a subject with its actions. Logs need to be regularly reviewed and maintained in a secure and consistent manner, otherwise are not admissible as evidence. *Automated reports*, based on certain predefined criteria or thresholds, known as *clipping levels*, are a reliable approach. For example, a clipping level may be set to generate a report when more than three failed logon attempts occur in a given period or at any attempt to use a disabled user account. These reports help a system administrator or security administrator to more easily identify possible break-in attempts. Other more intrusive approaches are possible, such as *keystroke monitoring* (*i.e.*, recording keystroke entries by a user during an active session), but are usually avoided due to their privacy implications.

1.1.1 Authentication

The most important access control function is authentication, which embodies any method of proving the identity of a subject. It is based on (a combination of):

- Something the user **knows** (password, pin)
- Something the user **has** (smart card, security token)
- Something the user **is/does** (fingerprint, retina, voice)
- **Where** the user is (company firewall, GPS proximity)

Something the user knows

The first method of proving a user's identity is to ask her for something she is expected to know. The typical example is a (*standard*) *password*, that is a (supposedly random) sequence of characters that the user is required to keep secret. The strength of a password depends on how difficult it is to guess it for an attacker. A password should be secure at least against the following attacks (from the weakest to the hardest):

- *Popular passwords*: only trying specific easy-to-remember passwords
- *Dictionary*: sequentially trying all passwords obtained combining “words” of a suitable dictionary
- *Brute force*: sequentially trying all possible passwords

This means that not only very popular passwords (*e.g.*, “12345678”, “password”) but also readable passwords (*e.g.*, “tennischampion”, “ilovemydog”) should be avoided, especially if closely related to personal but not so private details. A long, random-looking password is preferable, better if obtained from a large alphabet (*e.g.*, numbers + lower/upper case letters + symbols).

The main problem with standard password is the readability/memorability vs. strength/guessability trade-off: passwords should be secure but not to hard to remember. To solve the problem it is possible to rely on so-called *cognitive passwords*. A cognitive password requires a user to answer a question, presumably something they intrinsically know. They generally use facts ('what was your high school?') or opinions ('what is your favorite color?') based information. Fact based are usually easier to remember, but also easier to guess, while opinion based have better memorability/guessability ratios, but can be harder to remember (*e.g.*, an opinion can change over time).

An alternative approach, especially for sensitive applications (*e.g.*, home banking) consists in relying on so-called *one-time-passwords* (OTP), that are disposable passwords used only once and then thrown away. They are currently used only in selected contexts such as prepaid cards, or token devices – that generate the one time password for the user to submit to an authentication server. Tokens are increasingly used in different application settings, also thanks to the diffusion of handheld devices that can be used as token devices. However, it is important to keep in mind that token devices require synchronization between user and server or challenge/response protocols, that are not always feasible. Additionally, a token device is not really something you know, but rather something you have, as we will better discuss later.

Finally, it is possible to use sequences of characters longer than a password – that is, a *passphrase*. Usually a user enters this phrase into an application which transforms the value into a virtual password. Even if longer than a standard password, it must be kept in mind that passphrases may be very weak, due to the low entropy of expressions in natural language (a meaningful sentence has a way more predictable structure than a random string) and the typical presence of cultural references (movies, sports, etc.) that are easy to guess.

Other than the previously mentioned categories of attacks to standard passwords (brute force, dictionary, and popular passwords attacks) it is worth to mention a few other attack vectors:

- *Electronic monitoring*: listening to network traffic to capture information, especially when a user is sending her password to an authentication server;

electronic monitoring includes copying and reusing the password at another time, known as replay attack

- *Social engineering*: an attacker psychologically manipulates a user into performing actions and/or divulging confidential information, often relying on social media
- *Phishing*: tricking the user into inserting her private credentials to access fake resources
- *Workstation hijacking*: tampering with the user's workstation to obtain authorization through automatic/saved authentication
- *Access the password file*: consists in stealing the password file from the authentication server, thus obtaining many users' private credentials

The success likelihood of all the above attacks somehow depends on the behavior of the user: brute force and dictionary attacks succeed against badly chosen passwords, while other attacks, such as social engineering and phishing, work if the user is inexperienced or careless. In all cases, to establish clear rules and policies that all users *must* follow is highly recommended to boost the security of a system. These rules may include:

- Choosing a password that respects a precise format (*e.g.*, at least 10 characters, including at least two numbers and at least 1 special symbol)
- Changing the password at regular intervals
- Never answer to emails received from an unknown sender
- Do not share any work-related information on social networks

The possibility that an attacker steals the password file is instead independent of the behavior of the users, but it can only be caused by security weaknesses of the authentication server. So-called *password hashing*, depicted in Figure 1.2, is the standard solution to significantly limit the impact of a similar attack. The rationale is that the server should store an *obfuscated* list of passwords, instead of the password themselves. Recurring to cryptography, other than affecting the efficiency of the whole authentication mechanism, would pose the problem of how to securely store the secret key. (Cryptographic) Hash functions are *one-way* functions for which it is computationally infeasible to find either a pre-image or a collision, so that knowing the hash *digest* of a password yields no information about the password itself. So-called *salt* is a pseudo-random string added to avoid attacks based on heavy pre-computation campaigns.

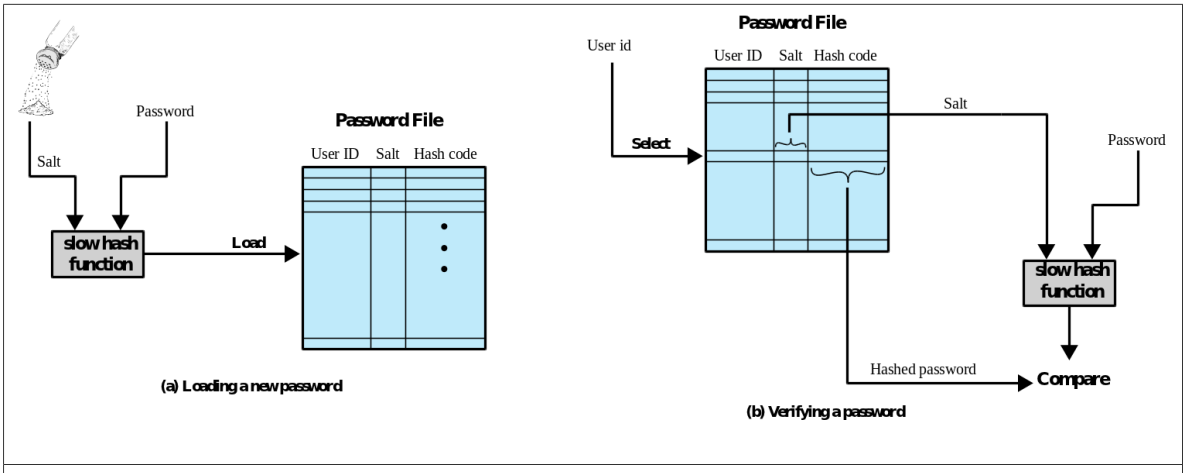

Figure 1.2 – Password hashing

Even when the password file is *hashed*, a few attacks are possible:

- *Dictionary attacks*: developing a large dictionary of possible passwords and trying each against the password file; each password must be hashed using each salt value and then compared to stored hash values
- *Rainbow table attacks*: pre-computing tables of hash values for all salts, resulting in huge table of hash values, but in a faster attack; it can be countered by using a sufficiently large salt value and a sufficiently large hash length
- *John the Ripper*: a famous open-source password cracker first developed in in 1996, it uses a combination of brute-force and dictionary techniques

Something the user has

Another approach to verify someone's identity consists in requiring possession of something that only her is supposed to have. Very common examples include keys, documents, memory cards, smartphones, or other similar devices.

A *token device*, or password generator, is a handheld device that has an LCD display and possibly a keypad, separate from the computer the user is attempting to access, and that generates a one-time-password on demand. The main benefit of token devices is their ability to provide two-factor authentication without being vulnerable to electronic eavesdropping (*e.g.*, wiretapping or packet analysis). The main limitations are their exposure to human errors, their reliance on battery power, and the requirement for synchronization (time or counter) with the server or for a challenge/response mechanism. To make such devices more user-friendly and reliable, they are nowadays often implemented through smartphone apps.

A *memory card* holds information but is unable to process data. It is generally used in combination with a password or a PIN to boost security. The most common

memory cards are magnetic stripe cards. Memory cards are mostly used as electronic keys for physical access (*e.g.*, for hotels and ATMs).

Differently from memory cards, *smart cards* can both hold and process information. Card readers provide the required energy, either through physical contact or contactless, and a PIN or a password is used to unlock smart card functionalities. Smart cards are usually blocked after a number of failed login attempts. The main use cases are credit cards, SIM cards, electronic Ids and pay tv.

Memory cards and smart cards are susceptible to a number of attacks, mostly based on interfering with the hardware or the software components of the card. The most diffused are:

- *Fault generation*: using abnormal environmental conditions to generate malfunctions in the system/reader
- *Microprobing*: accessing the chip surface directly, to observe, manipulate, and interfere with the device
- *Side Channel Attacks:* nonintrusive techniques consisting in monitoring the analog characteristics of power supply and interface connections and any electromagnetic radiation, and correlate the measured patterns with secret data stored on the card
- *Software attacks*: using the normal communication interface and exploiting security vulnerabilities found in the protocols, cryptographic algorithms, or their implementation

From the user's perspective, it is fundamental to always make sure that the card is safely stored and that readers have not been tampered with.

Something the user is/does

A 'Special case' of asking for something the user has consists in relying on unique personal attributes, mostly biometrics, *i.e.*, individual's unique physical characteristics. The most used biometrics are fingerprints, retina scan, iris scan, hand geometry, facial scan and voice recognition. Figure 1.3 shows three aspects of biometric scanning systems that are independent of the specific biometric used: enrollment of the user's biometric into the database, verification of a claimed identity, and identification of an unknown user.

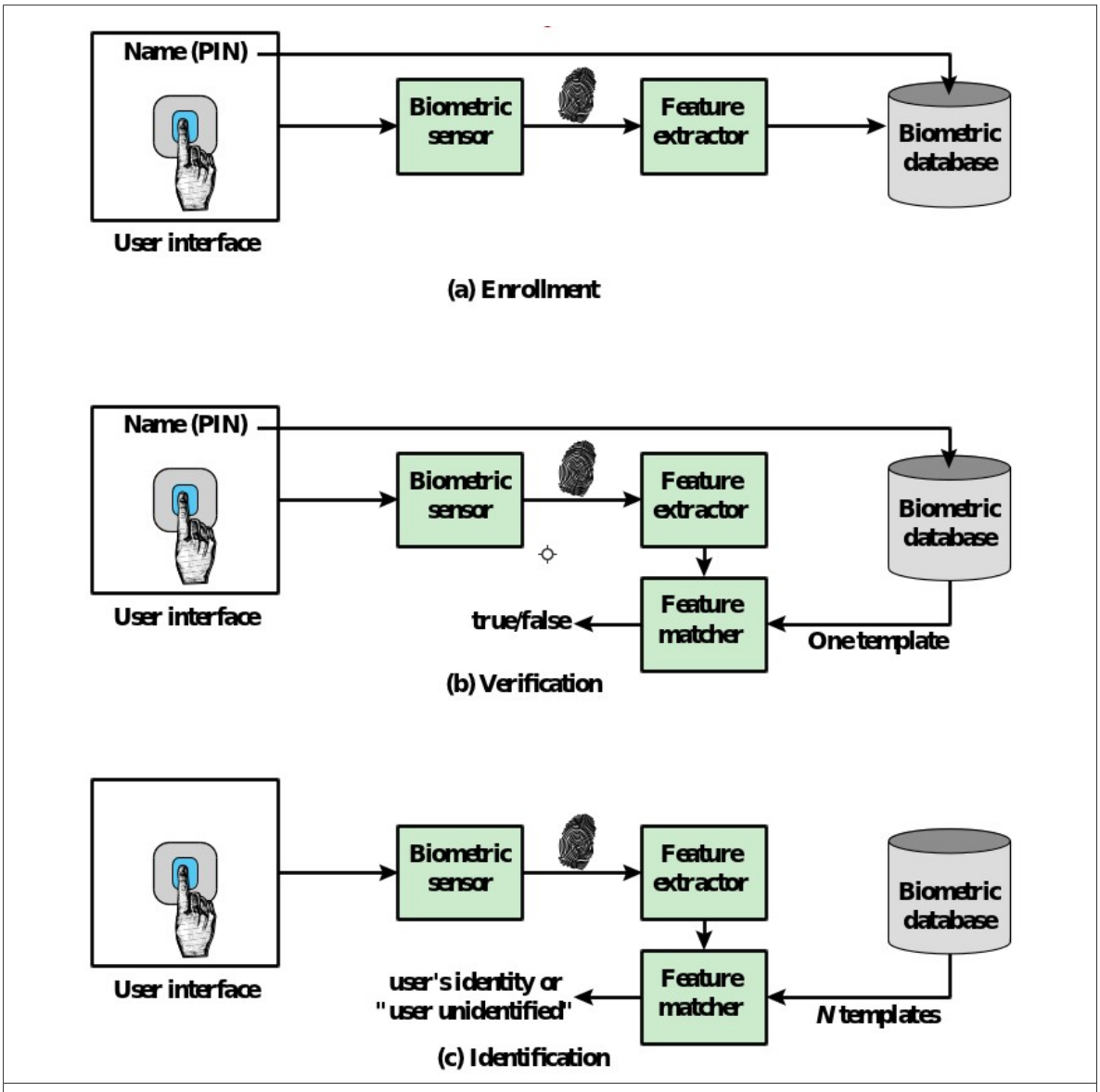

Figure 1.3 – Biometric scanning systems

Since every person's *fingerprint* is unique, fingerprint analysis is a very affordable and convenient method of verifying a person's identity. The lines that create a fingerprint pattern are called 'ridges' and the spaces between ridges are called 'valleys'. The recognition is based on patterns of distinctions in ridges and valleys.

Retinal scan technology maps the capillary pattern of the retina (a very thin nerve on the back of the eye). *Iris scans* technology maps marks and shape of iris. Retina and iris scans have the highest accuracy, but people are usually hesitant to use retina/iris scanners and authentication is 'slow' (a few seconds).

Hand and *facial scans* are based on measuring specific characteristics of a person's hand or face. They take a number (~100) of measurements of the length, width, thickness, and surface area of a person's hand (and fingers) or face. Such

scans are typically faster than retina/iris scans, but they require larger databases and need updates because the geometry of hands and faces changes over time.

To select the most suitable biometric system, it is useful to keep in mind two different aspects:

- The trade-off between accuracy and cost, depicted in Figure 1.4, that synthetizes the value-for-money of different biometrics
- The trade-off between false-match and false-nonmatch rates, depicted in Figure 1.5, that describes how prone to errors different biometric scans are

The figures show why fingerprints are the most diffused biometric: they have a very reasonable cost/accuracy ratio, and false positive can be reduced significanty without causing too many false negatives.

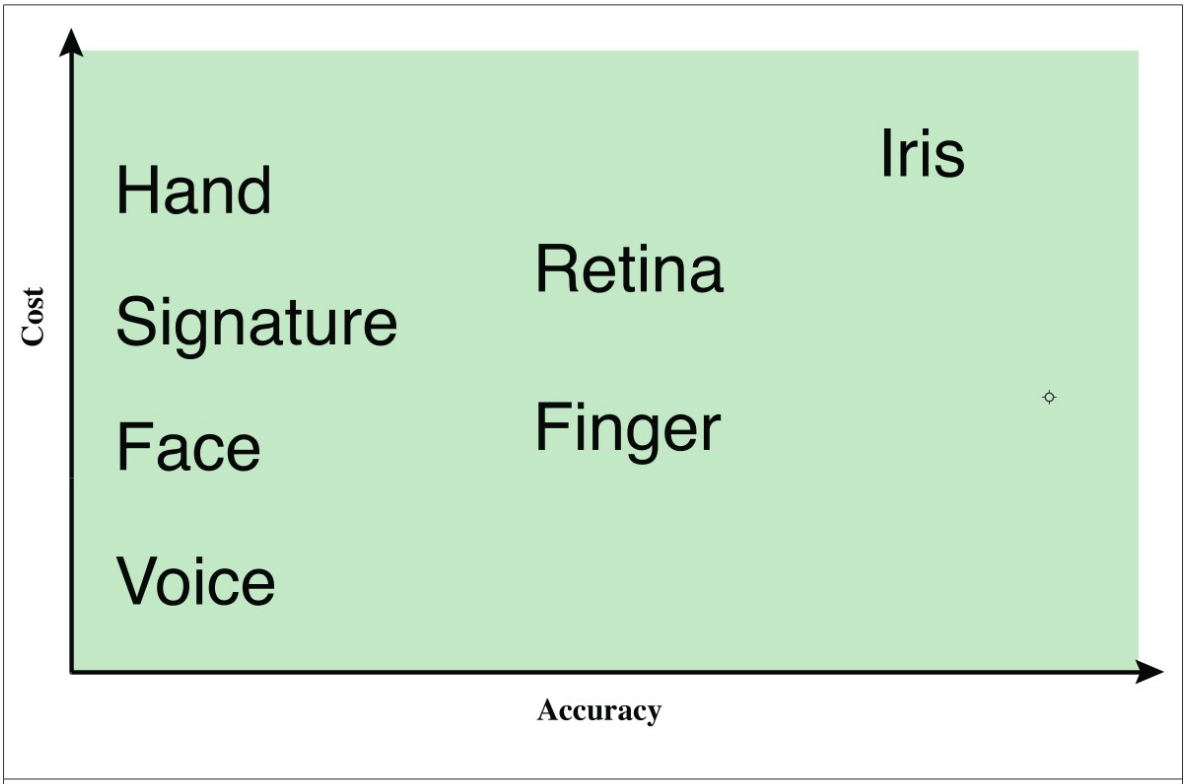

Figure 1.4 – The trade-off between accuracy and cost for biometrics

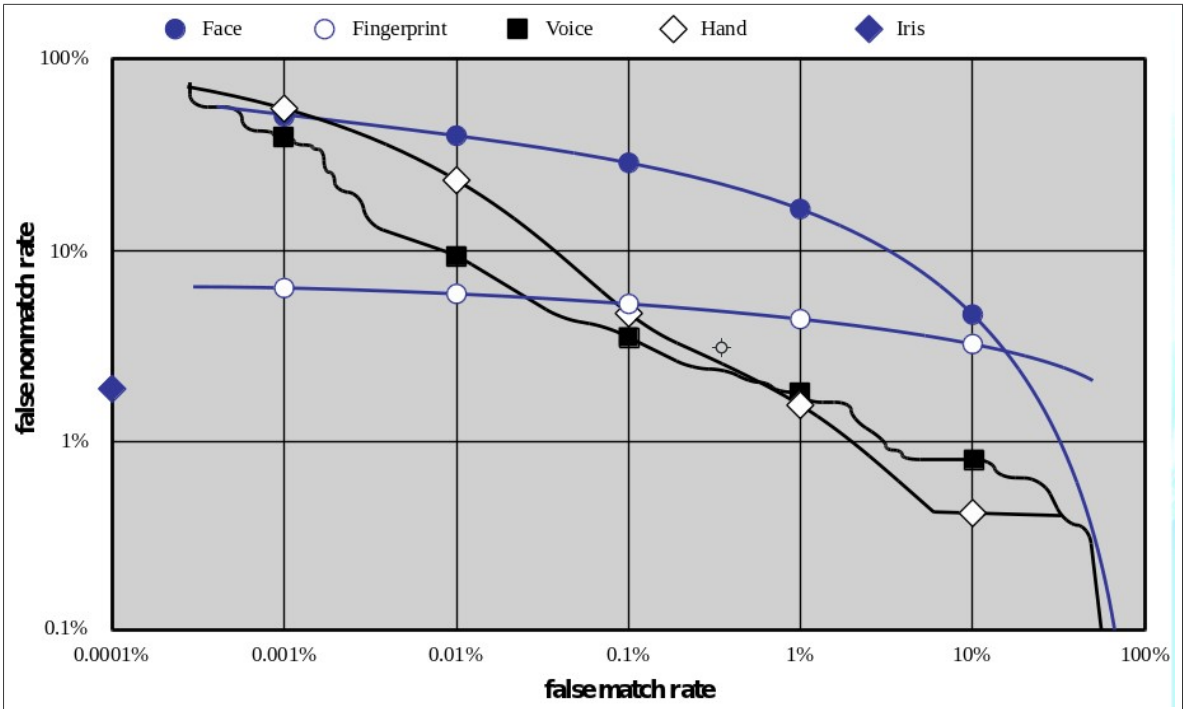

Figure 1.5 – The trade-off between false-match and false-nonmatch rates for biometrics

Remote authentication

Authentication over a network, the Internet, or a communications link is more complex due to additional security threats such as communication eavesdropping, password capturing, replay attacks (*i.e.*, replaying an authentication sequence that has been observed). Generally, remote authentication relies on some form of challenge-response protocols to counter threats, which means that upon receiving an authentication request the authentication server sends a “question” to the user and grants or denies access based on the answer.

A widely used approach to remote authentication relies on the so-called *Message Authentication Code* (MAC), depicted in Figure 1.6: some cryptographic algorithm is used to produce a short tag, the MAC, for a specific message (*e.g.*, the challenge), based on a secret key that only the client (and possibly the server) is supposed to know.

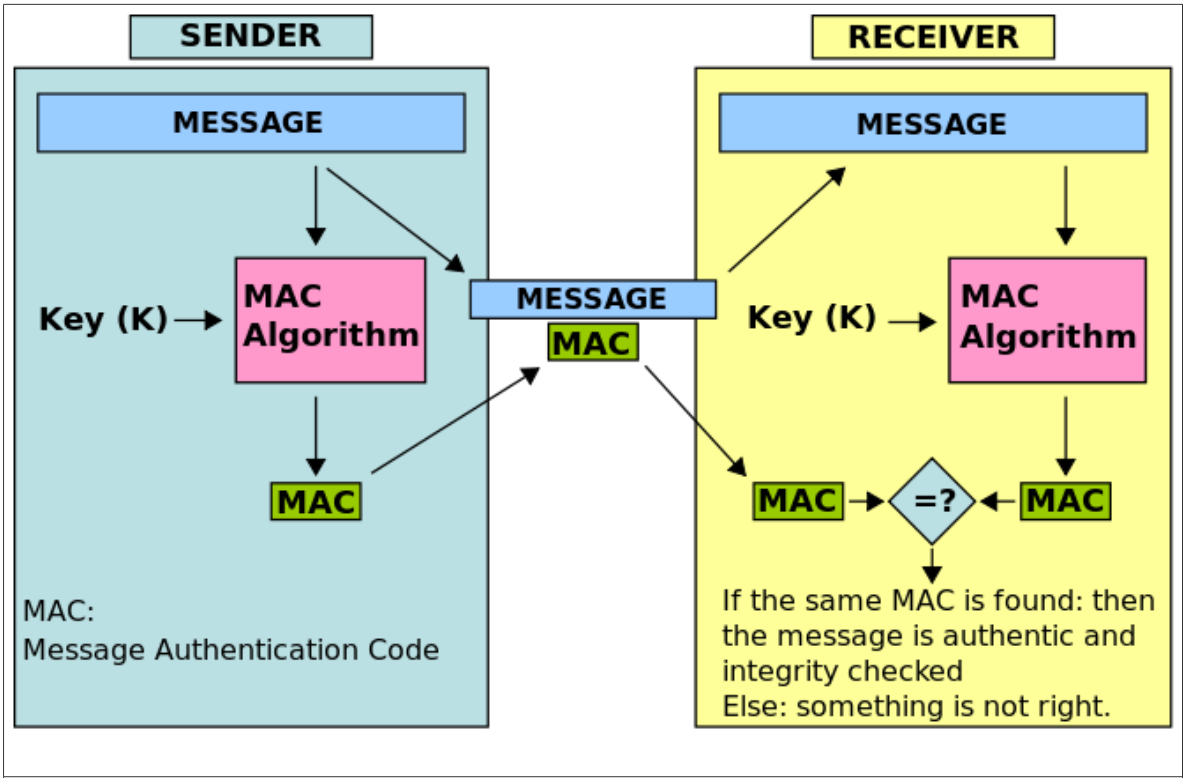

Figure 1.6 – Message Authentication Code

The most diffused attacks to a remote authentication system are:

- *Denial-of-Service* (DoS): attempting to disable a user authentication service by flooding the service with numerous authentication attempts
- *Eavesdropping*: attempting to learn the password by some sort of attack that involves the physical proximity of user and adversary
- *Host Attacks*: directed at the user file at the host where passwords, token passcodes, or biometric templates are stored
- *Trojan Horse*: an application or physical device masquerading as an authentic application or device for the purpose of capturing a user password, passcode, or biometric
- *Client Attacks*: attempting to achieve user authentication without access to the remote host or the intervening communications path
- *Replay*: repeating a previously captured user response

1.2 CATEGORIES of ACCESS CONTROL

Access control can be categorized based on two main aspects, the type of control and its purpose.

There are three main types of access control:

- **Administrative** controls define roles, responsibilities and policies
- **Technical** controls use hardware/software technologies to implement access control
- **Physical** controls ensure safety and security of the physical environment

More precisely, administrative controls ensure that technical and physical controls are understood and properly implemented, technical controls consist in designing and implementing technological solutions to enforce specific functionalities, while physical controls are more 'traditional' solutions such as HVAC (Heating, Ventilating and Air Conditioning), fences and motion detectors.

According to its purpose, an access control technique can be categorized as:

- *Preventive* if it aims at avoiding an incident
- *Deterrent* if it aims at discouraging an incident
- *Detective* if it aims at identifying an incident
- *Corrective* if it aims at remedying/mitigating the damage and restore controls
- *Recovery* if it aims at restoring normal conditions
- *Compensating* if it aims at implementing alternative control

A secure information system should implements access control techniques of all type and with all purposes. However, preventive and detective controls are the most important to avoid incidents and limit their impact. Typical examples of preventive controls are:

- Pre-employment background checks (administrative)
- Data classification and labeling (administrative)
- Security awareness (administrative)
- Separation of duties (administrative)
- Passwords (technical)
- Biometrics (technical)
- Firewalls (technical)
- Anti-virus (technical)
- Badges (physical)

- CCTV (physical)
- Locking computer cases (physical)
- Disabling USB port (physical)

Typical examples of detective controls are the following:

- Job rotation (administrative)
- Inspections (administrative)
- Use of auditors (administrative)
- Reviewing audit logs (technical)
- Reviewing violations of clipping levels (technical)
- Forensics (technical)
- Intrusion detectors (physical)
- Video cameras (physical)
- Guards responding to an alarm (physical)

1.3 ENTERPRISE-WIDE SOLUTIONS – SINGLE SIGN-ON

To identify a suitable *enterprise-wide* solution for access control means to choose the preferred access control model, to select and implement different access control technologies, and to deal with consequent requirements, such as:

- User provisioning
- Password synchronization and reset
- (Centralized) auditing and reporting
- Integrated workflow (increase in productivity)
- Regulatory compliance

Access control administration comes in three basic forms:

- **Centralized**
- **Decentralized**
- **Singe Sign-On**

Decentralized solutions give control of access to the people who are closer to the resources. They have no methods for consistent control, so they lack proper consistency, but similar solution have no 'hidden' flow of information and have more control on privileges.

In *centralized solutions* one entity is responsible for overseeing access to all corporate resources. It provides a consistent and uniform method of controlling access rights. Protocols are agreed upon ways of communication, and Attribute Value Pairs are used to define fields that only accept specific range of values. Centralized solutions solve two major problems for users:

- Users don’t need to remember multiple sets of authentication credentials
- The applications they are logging into can't share user data

Yet, once a user has logged into App 1, logging into App 2 doesn’t feel automatic: even though the required credentials are identical, the user would still need to enter her authentication information again.

Finally, *Single Sign-On* (SSO) denotes a diffused access control model for multiple, related, but independent software systems. Users log in once and gain access to all systems without being prompted to log in again at each of them. A Single sign-off action is required to terminates access to multiple software systems. Since different applications and resources support different authentication mechanisms, SSO must translate and store different credentials compared to what is used for initial authentication. SSO solves two major problems for users:

- They don’t need to enter authentication information multiple times
- They don't need to remember multiple sets of login credentials

Once a user has logged into App 1, logging into App 2 works as follows: App 2 simply determines whether it can authenticate the user based on information the SSO Identity Provider (IDP) provides.

To better understand the differences among the three approaches, let us consider the following example: a user logs on to her workstation, then decides to access a company database.

- Decentralized: the database requires another username and password for authentication
- Centralized: the user has to enter her authentication information again, but the required credentials would be identical to the credentials she used to log on to her workstation
- SSO: the database determines whether it can authenticate the user based on information the network's authentication server provides

1.3.1 Kerberos

Kerberos is the most known computer network authentication protocol for SSO. It has the following properties:

- It allows nodes communicating over a *non-secure network* to prove their identity to one another in a secure manner

- It is designed for a client–server model: it provides *mutual authentication* – both the user and the server verify each other's identity
- Protocol messages are *protected* against *eavesdropping* and *replay* attacks
- The security of the protocol relies heavily on participants being *loosely synchronized* and on short-lived assertions of authenticity called *Kerberos tickets*
- It is based on the *Needham-Schroeder* symmetric-key protocol:
    - It makes use of a *key distribution center* (KDC), which consists of an *Authentication Server* (AS) and a *Ticket Granting Server* (TGS)
    - The KDC maintains a database of secret keys; each entity on the network (*e.g.*, client or server) shares a secret key known only to itself and to the KDC
    - For communication between two entities, the KDC generates a session key which they can use to secure their interactions

The operating principles of Kerberos, depicted in Figure 1.7, can be summarized as follows:

- The client authenticates itself to the Authentication Server and receives a ticket (all tickets are time-stamped)
- The client contacts the Ticket Granting Server; using the ticket it demonstrates its identity and asks for a service
- If the client is eligible for the service, then the Ticket Granting Server sends another ticket to the client
- The client then contacts the Service Server, and using this ticket it proves that it has been approved to receive the service

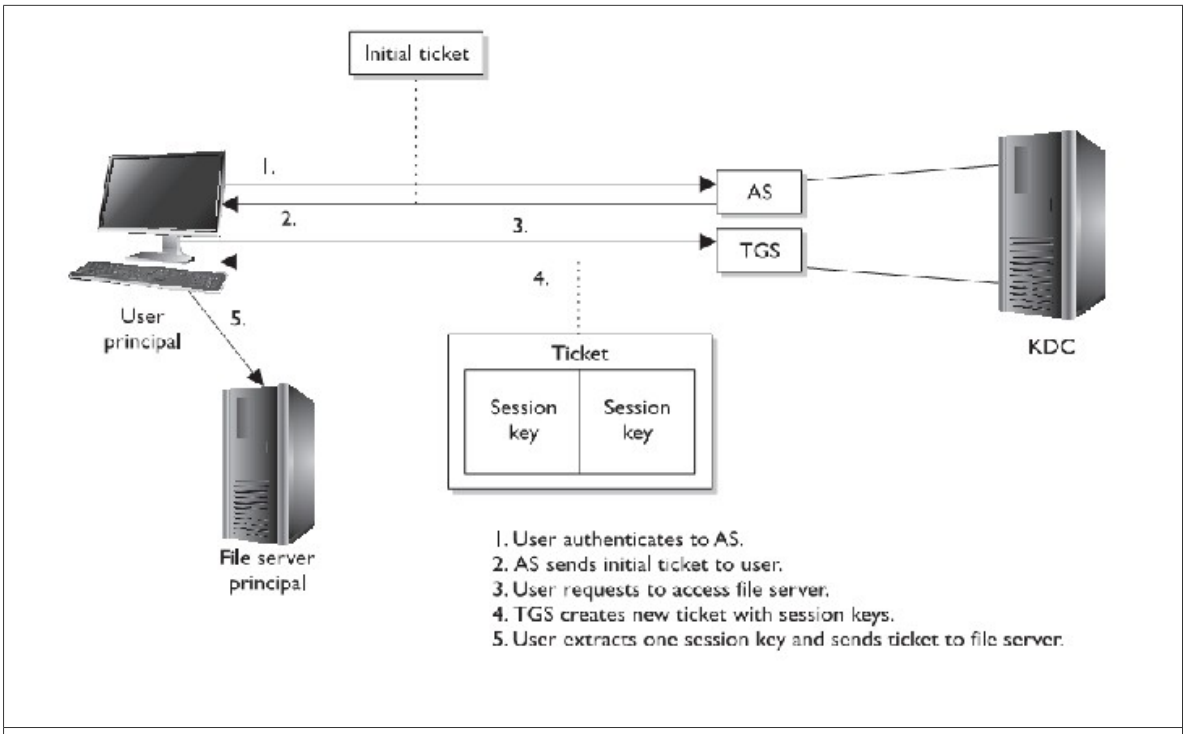

Figure 1.7 – Kerberos

Kerberos has a few drawbacks, that it shares with most other SSO solutions:

- *Single point of failure*: Kerberos requires continuous availability of a central server; when the server is down, no one can log in; this limitation can be mitigated by using multiple Kerberos servers and fallback authentication mechanisms
- Kerberos requires the clocks of the involved hosts to be *synchronized*: tickets have a time availability period and if the host clock is not synchronized with the Kerberos server clock, the authentication fails; default configuration requires that clock times are no more than five minutes apart - in practice, Network Time Protocol daemons are usually used to keep the host clocks synchronized
- The administration protocol is *not standardized* and differs between server implementations
- Since all authentication is controlled by a centralized KDC, compromise of this authentication infrastructure will allow an attacker to *impersonate any user*

1.4 ACCESS CONTROL MODELS

An *access control model* defines the general principles that determine how access rights to objects are granted to subjects. The most diffused models can be organized into a tree, depicted in Figure 1.8, according to *who* is responsible of determining access and on *how* access is determined.

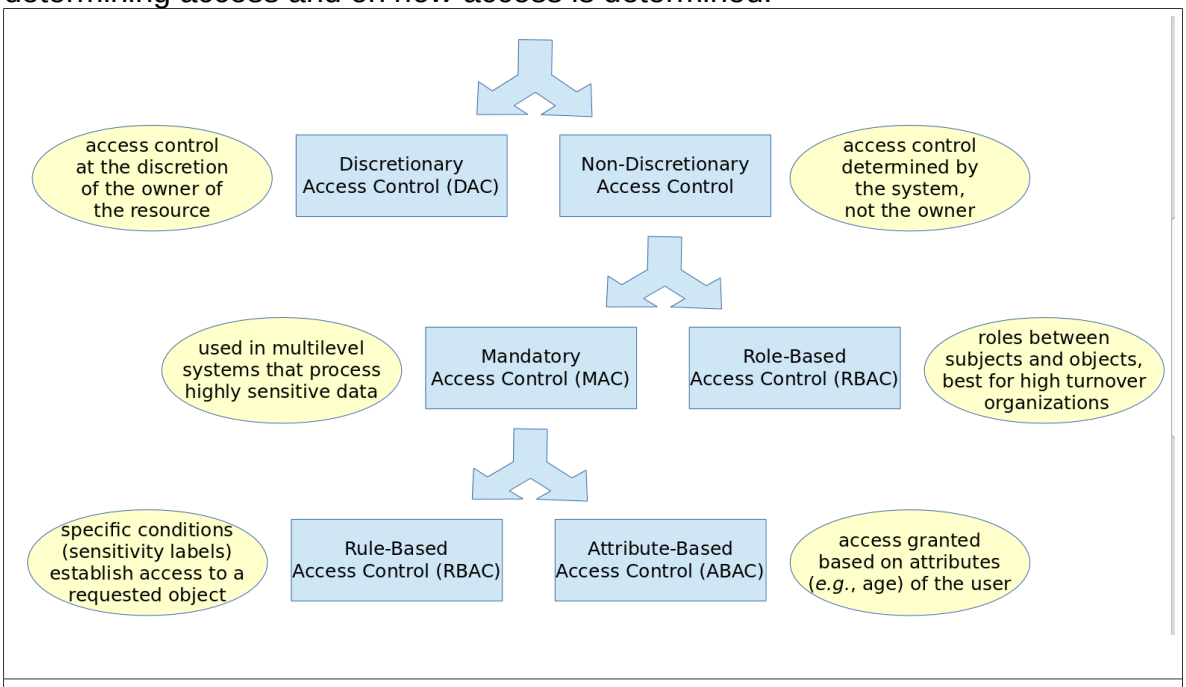

Figure 1.8 – The most diffused access control models

1.4.1 Discretionary Access Control

In *Discretionary Access Control* (DAC), access control is at the discretion of the owner of the resource, who specifies which subjects can access that resource and what privileges they have. DAC relies on two important concepts:

- *File and data ownership*: every object in the system has an owner. In most DAC systems, each object's initial owner is the subject that caused it to be created. The access policy for an object is determined by its owner
- *Access rights and permissions*: these are the controls that an owner can assign to other subjects for specific resources

DAC can be implemented through:

- *Access Control Lists* (ACL): an ACL is a list of permissions attached to an object, that specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects. Each entry in a typical ACL specifies a subject and an operation

- *Capability List* (CL): a CL is similar to an ACL, except that the list is attached to a subject, and specifies the access rights a certain subject possesses pertaining to specific objects
- *Access Control Matrices* (ACM): an ACM is a matrix with a row per subject and a column per object, in which the entry in position *i,j* establishes the rights of subject *i* with respect to object *j*

The operating principles of these two structures are vey similar, but ACL are usually preferred because in most systems most entries of an ACM would be empty, so that ACL are more efficient.

DAC is called *strict* if the owner is the only one who has discretionary authority to grant access to an object and that ownership cannot be transferred. It is instead called *liberal* when the owner is allowed delegating discretionary authority for granting access to an object to other users. Liberal DAC comes into multiple flavors:

- One Level Grant: The owner can delegate grant authority to other users but they cannot further delegate this power
- Two Level Grant: In addition to a one-level grant, the owner can allow some users to further delegate grant authority to other users
- Multilevel Grant

Finally, DAC with Change of Ownership allows a user to transfer ownership of an object to another user.

1.4.2 Mandatory Access Control

In *Mandatory Access Control* (MAC), access control is determined by the system instead of the owner of the resource. MAC is typically used in multilevel systems that process highly sensitive data, such as classified government and military information. A multilevel system is a single computer system that handles multiple classification levels between subjects and objects.

In a MAC-based system, all subjects and objects must have *sensitivity labels* assigned to them:

- A subject's sensitivity label specifies its level of trust
- An object's sensitivity label specifies the level of trust required for access

In order to access a given object, the subject must have a sensitivity level equal to or higher than the requested object.

Controlling the import of information from other systems and the export to other systems (including devices such as printers) is a critical function of MAC-based systems. Sensitivity labels must be properly maintained and implemented so that sensitive information is appropriately protected at all times.

Figure 1.9 and Figure 1.10 synthetize the main differences between DAC and MAC, for what concerns, respectively, functioning and security.

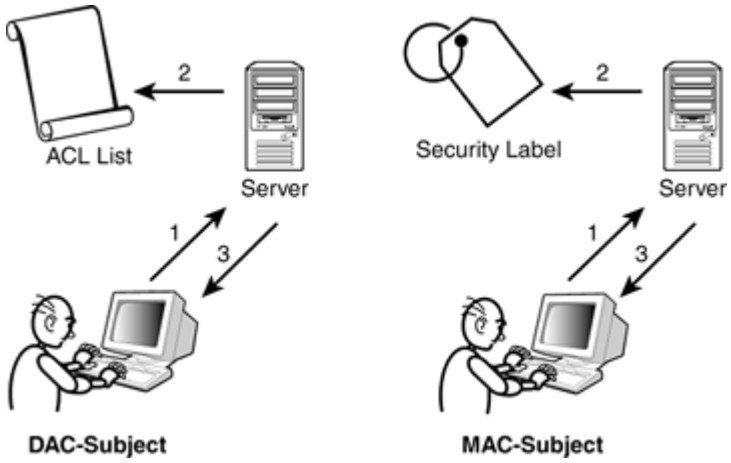

Figure 1.9 – DAC *vs.* MAC functioning

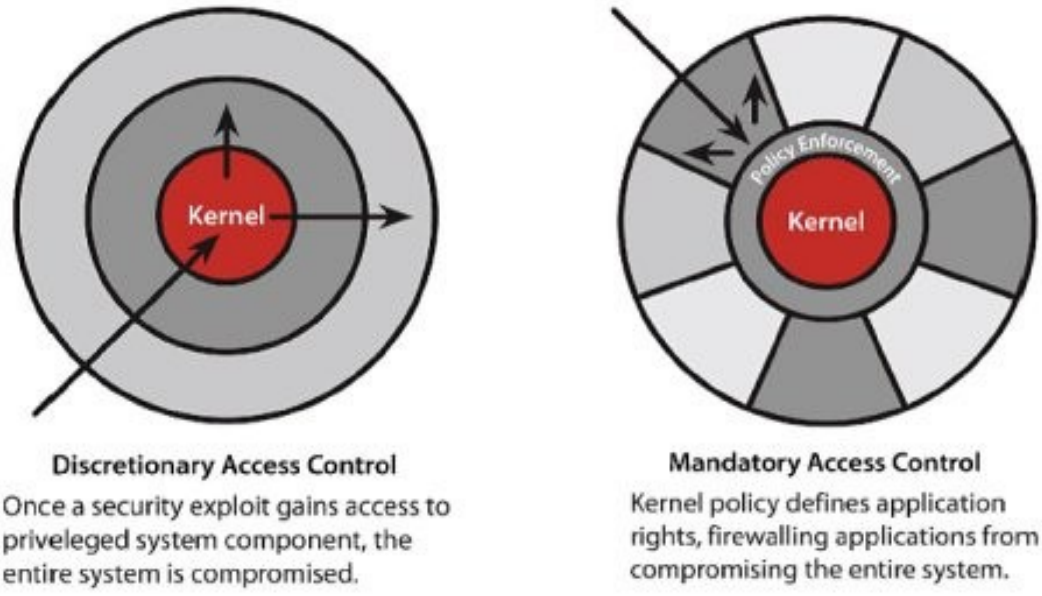

Figure 1.10 – DAC *vs.* MAC security

1.4.3 Rule-Based Access Control

In *Rule-Based Access Control* (RuBAC), specific conditions establish access to a requested object. All MAC-based systems implement a simple form of rule-based

access control to determine whether access should be granted or denied by matching an object's sensitivity label with a subject's sensitivity label.

Sensitivity labels can be of different types, for instance:

- Classification level, *e.g.*, Secret, Top secret, Confidential
- Category, *e.g.*, Information warfare, Treasury, Finance

Figure 1.11 shows an example of a RuBAC system based on classification levels.

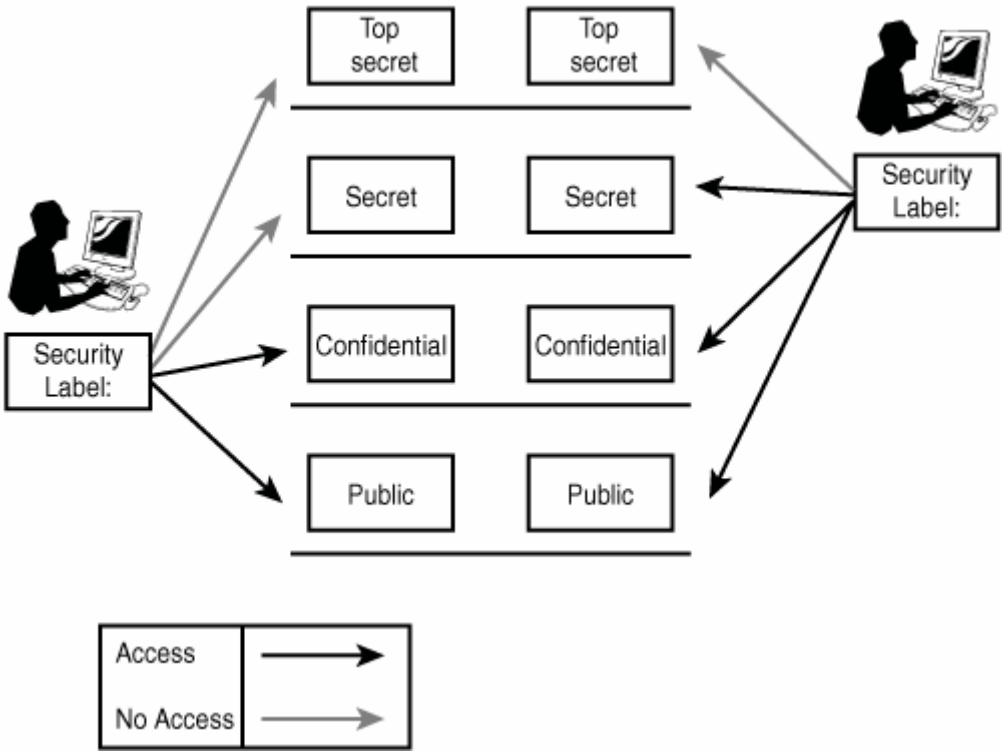

Figure 1.11 – A RuBAC system based on classification levels

RuBAC is often based on lattices and called *Lattice-Based Access Control* (LBAC). In fact, a lattice model is a particular mathematical structure (often representable as a graph) that defines greatest lower-bound and least upper-bound values for a pair of elements, such as a subject and an object. LBAC is used for complex access control decisions involving multiple objects and/or subjects.

1.4.4 Attribute-Based Access Control

In *Attribute-Based Access Control* (ABAC), access is granted not based on the rights of the subject associated with a user after authentication, but based on attributes of the user: the user has to prove so called claims about his attributes to the access control engine. An attribute-based access control policy specifies which claims need to be satisfied in order to grant access to an object. For instance the

claim could be "older than 18": any user that can prove this claim is granted access.

Users can be anonymous as authentication and identification are not strictly required. However, in general means for proving claims anonymously are necessary. This can for instance be achieved using anonymous credentials.

The general framework of an ABAC system is depicted in Figure 1.12.

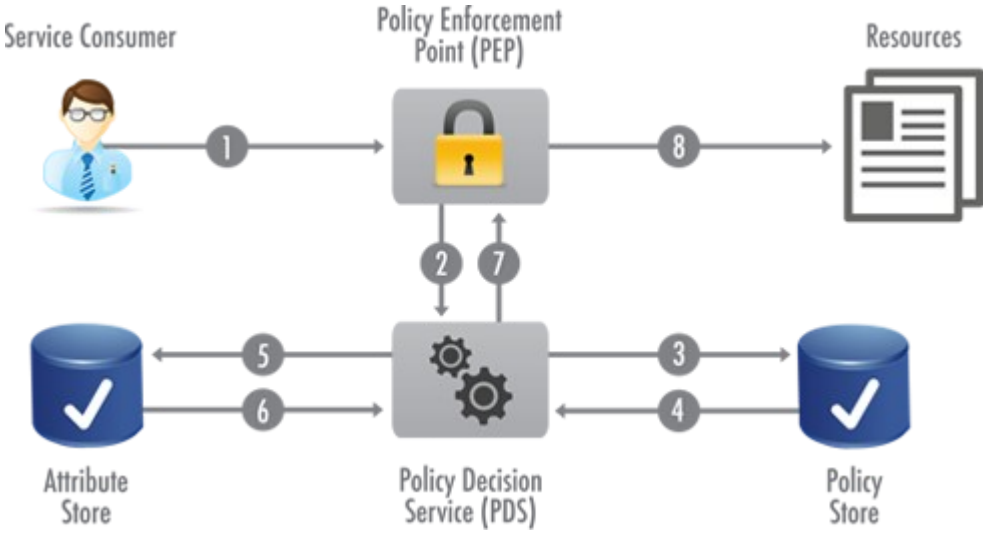

Figure 1.12 – The general framework of an ABAC system

1.4.5 Role-Based Access Control

Role-Based Access Control (RBAC) is used in both commercial applications and military systems, provided that multi-level security requirements exist. RBAC differs from DAC in that DAC allows users to control access to their resources, while in RBAC, access is controlled at the system level, outside of the user's control. However, although RBAC is an access policy determined by the system, not the owner, and is therefore non-discretionary, it can be distinguished from MAC primarily in the way permissions are handled: MAC controls read and write permissions based on a user's clearance level and possibly on additional labels, while RBAC controls collections of permissions that may include complex operations such as an e-commerce transaction, or may be as simple as read or write.

A role in RBAC can be viewed as a set of permissions: subjects are mapped to roles, while privileges are assigned to roles instead of subject. This structure is the best solution for an organization that has a high turnover, because the turnover only requires assigning proper roles to the users, without modifying the roles-to-permissions assignment.

There are three primary rules that define the functioning of RBAC systems:

- *Role assignment*: A subject can execute a transaction only if the subject has selected or been assigned a role
- *Role authorization*: A subject's active role must be authorized for the subject. Given the previous rule, this rule ensures that users can take on only roles for which they are authorized
- *Transaction authorization*: A subject can execute a transaction only if the transaction is authorized for the subject's active role. Given the previous rules, this rule ensures that users can execute only transactions for which they are authorized

Additional constraints may be applied as well, and roles can be combined in a hierarchy where higher-level roles subsume permissions owned by sub-roles. Most IT vendors offer RBAC in one or more products.

The subject-to-role-to-object assignment at the base of RBAC solutions is depicted in Figure 1.13.

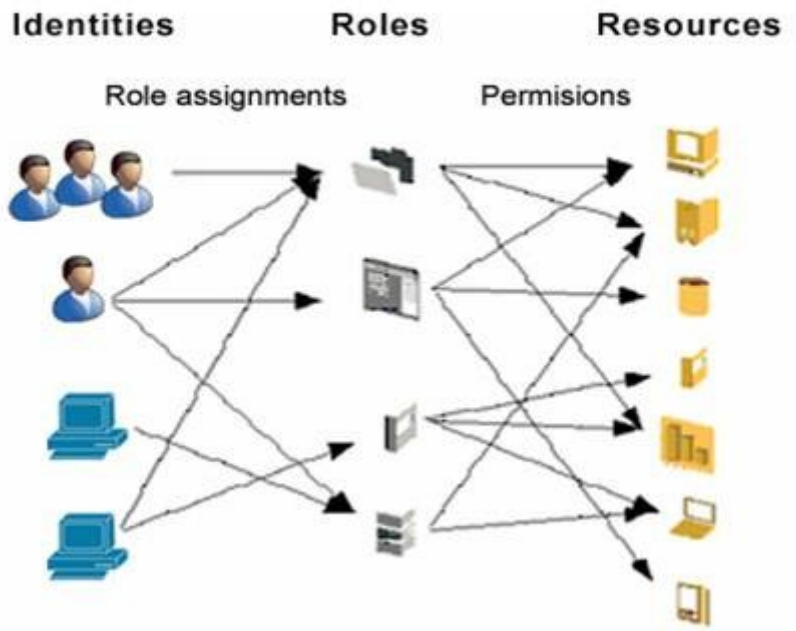

Figure 1.13 – The subject-to-role-to-object assignment in RBAC systems

RBAC is very interesting and it represent a challenging research area, in particular with respect to a non-trivial open problem: is there an optimal/perfect choice of roles for an organization?

1.4.6 Access Control Models Case-Study

To clarify the basic principles of and differences between the access control models seen so far, let us consider the following case study: Alice is an assistant to the CEO, while Bob is an assistant in the marketing department. Even if they have similar jobs, Alice may need specific permissions that Bob doesn’t need.

- DAC: According to DAC, for each resource, the owner decides what rights to assign to Alice and Bob. Default rights may exist and different rights are assigned explicitly
- MAC: In MAC, Alice and Bob are assigned labels (*e.g.*, "confidential" and "HR" to Alice, "public" and "marketing" for Bob) and they can access resources based on the resources' labels and on the company policy
- RBAC: In RBAC, two different roles with different permissions are created:
  - CEO's assistant
  - Marketing assistant

Alice is assigned to the "CEO's assistant" role, Bob to the "marketing assistant" role. This assignment is more fine-grained than MAC but less than DAC. However, possible problems may emerge, such as finding the needed number of roles, or understanding whether "CEO's assistant" should be a superclass of "marketing assistant".

1.4.7 Other Access Control Models

There are a few other less used access control models besides the ones mentioned so far:

- *User-Based Access Control* (UBAC) (or Identity-based): a system administrator defines permissions for each user, based on the individual’s needs. A similar approach potentially provides more finely grained permissions, but is too labor intensive to be effective
- *Content-Dependent Access Control* (CDAC): access depends on attributes of the resource. It is primarily used to protect databases containing potentially sensitive data. For example, a nurse may have access to blood tests unless the blood test is an HIV test (the system has to check which test it is in order to determine if the access is allowed)
- *Context-Based Access Control* (CBAC): access grants do not depend solely on who the user is and which resource it is, but also on the sequence of events that preceded the access attempt. For example, most firewalls implement "stateful inspection" by updating the "state" of every connection when a new packet arrives, and dropping packets or allowing them to continue, based not only on their content, but also on the current state, i.e., the context in which the packets arrive

Technologies to implement models

The following solutions are the most widely used to implement a specific access control model:

Access Control Matrix (ACM): a matrix with an entry for each subject-object pair, specifying rights of that subject over that object

Access Control List (ACL): a list that specifies all subjects that are authorized to access a specific object, with the corresponding access type

Capability List (CL): it specifies the access rights a certain subject possesses pertaining to specific objects

Constrained User Interface: it restricts users to specific functions by not allowing them to request functions or services beyond their privileges or role. For instance, a constrained interface can limit the available menus, data views, encryption, ...

1.5 ACCESS CONTROL POLICIES

Access control policies are fundamental instruments to partition system states into *authorized* (secure) states, that the system can safely enter, and *unauthorized* (non-secure) states, that identify the occurrence of a security breach. A secure system is a system that starts in an authorized state and never enters an unauthorized state.

To clarify the importance of policies, let us consider the following case study: a computer science class requires that students complete some homework on the computers of the department's lab; Anne forgets to read-protect her homework file, Bill notices it and copies Anne's homework; the University policy disallows cheating, including copying homework, with or without permission. Who cheated? Anne, Bill, or both?

- Bill surely cheated: The system has entered an unauthorized state (Bill having a copy of Anne’s assignment) because Bill did not respect the policy that forbids copying homework assignments
- Anne did not protect her homework, but is it required by any security policy? If not, she cannot be considered responsible of a security breach
- Is it the Department’s fault? A more secure policy should have explicitly required read-protect. Alternatively, automatic read-protect could have been implemented.

1.5.1 Policy Models

It is often useful to rely on abstract descriptions of a policy or a class of policies, called *policy models*. Policy models guarantee multilevel security, prohibiting direct or indirect information flow through a control of access priviliges and inter-process communication channels. The aspects of a system controlled by a policy model are depicted in Figure 1.14.

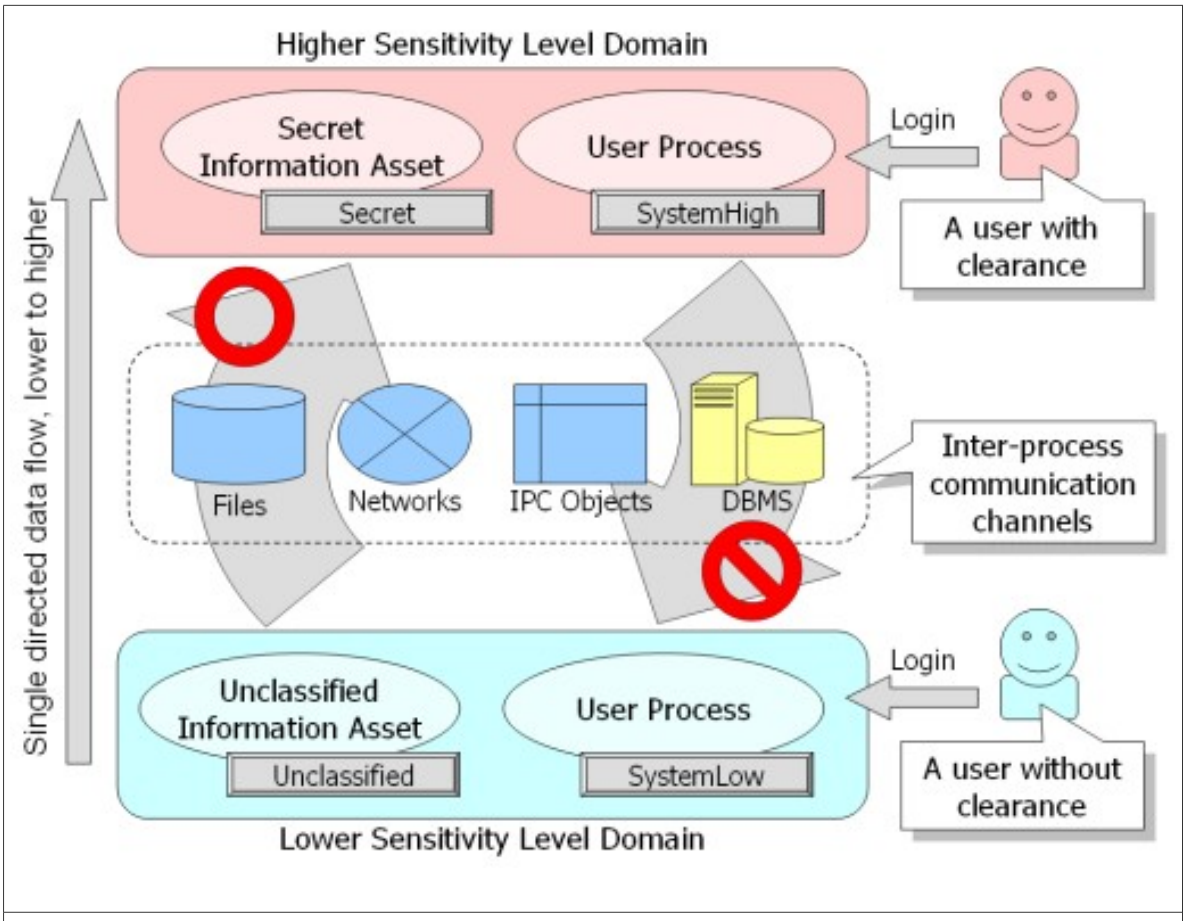

Figure 1.14 - The aspects of a system controlled by a policy model

Different models focus on different points of interest in policies. For instance, confidentiality-oriented models prescribes specific solutions to prevent users to read from objects with a higher sensitivity level and to write on objects with a lower sensitivity level, in order to avoid a flow of information that circumvents the sensitivity level assigned to an object. Conversely, integrity-oriented models aim at guaranteeing data integrity avoiding that users write on objects with a higher sensitivity level than their own or read from an object with a lower one. The idea in this case is to enforce integrity by preventing a flow of information from less to more reliable users/objects.

Some integrity policies use the notion of transaction (a series of actions):

- The system begins in a consistent state
- Only transactions from consistent state to consistent state are allowed
- Actions cannot be interrupted: if the actions are completed, the system is in consistent state, otherwise the system reverts to the last consistent state preceding the transaction

Specific languages have been introduced to express security policies in a precise way. In so-called high-level languages, policy constraints are expressed abstractly, independently of the enforcement mechanism. The constraints aim at restricting actions and entities but need to be unambiguous. For instance, consider a Web Browser:

- The goal of the policy is to restrict actions of Java programs that are downloaded and executed under the control of a web browser
- A high-level language specific to Java programs is the best choice to express constraints as conditions restricting creation of classes and invocation of entities, while being independent of the enforcement mechanism

Low-level languages, conversely, express policy constraints in terms of program options, input, or specific characteristics of entities on system. For instance, consider a X Window System (UNIX-based X11 Windowing System):

- Access to X11 display is controlled by a list that says what hosts are allowed or disallowed access
- *xhost +groucho -chico* means that connections from host groucho are allowed, while from host chico are not

Bell-LaPadula Confidentiality Policy

The goal of the Bell-LaPadula confidentiality model is to prevent the unauthorized disclosure of information. The main requirement is therefore to avoid information flow from trusted to untrusted entities. The operation principles are:

- Subjects and objects are labeled with security levels that form a partial ordering
- The model only considers information flows that occur when a subject observes or alters an object
- Access permissions are defined through an access control matrix and security levels
- The policy is NO READ-UP, NO WRITE-DOWN, so as to avoid information flow from ‘higher’ confidentiality levels down to ‘lower’ confidentiality level

Biba Integrity Policy

The goal of the Biba integrity model is to guarantee that information can be trusted. The main requirement is to avoid information flow from untrusted to trusted entities. The operation principles are:

- Subjects and objects are labeled with security levels that form a partial ordering

- The model only considers information flows that occur when a subject observes or alters an object
- Access permissions are defined through an access control matrix and security levels
- The policy is NO READ-DOWN, NO WRITE-UP, so as to avoid information flow from ‘lower’ integrity levels up to ‘higher’ integrity levels

Policies + RBAC: Case Study

Let us now consider a case study that shows a concurrent implementation of an access control policy and an RBAC access control model: RBAC96.

It is based on three sets of entities called users (U), roles (R), and permissions (P):

- A user is a human being or an autonomous agent
- A role is a job function or job title within the organization with some associated semantics regarding the authority and responsibility conferred on a member of the role
- A permission is an approval of a particular mode of access to one or more objects in the system

Additionally, RBAC96 has the following properties:

- It uses a role hierarchy RH, written as ≥
- It distinguishes (normal) roles and permissions from administrative roles and permissions
- It uses the concept of session (S)

More precisely, the following sets completely define the entities involved in RBAC96:

- U, a set of users
- R and AR, regular roles and administrative roles
- P and AP, regular permissions and administrative permissions
- S, a set of sessions
- PA $\subseteq$ P x R , a many-to-many permission to role assignment relation
- APA $\subseteq$ AP x AR, a many-to-many permission to administrative role assignment relation
- UA $\subseteq$ U x R , a many-to-many user to role assignment relation
- AUA $\subseteq$ U x AR, a many-to-many user to administrative role assignment relation
- RH $\subseteq$ R x R , a partially ordered role hierarchy

- ARH $\subseteq$ AR x AR, partially ordered administrative role hierarchy

The access control policy is expressed in terms of security labels attached to subjects and objects:

- A label on an object is called a security classification
- A label on a user is called a security clearance
- The read-down policy is expressed by the rule $\lambda(s) \geq \lambda(o)$
- The write-up policy is expressed by the rule $\lambda(s) \leq \lambda(o)$
- The write-equal policy is expressed by the rule $\lambda(s) = \lambda(o)$

1.6 KEY MANAGEMENT

As discussed before, one of the primary principles of access control is authentication. Authentication techniques require to obfuscate data, but everyone knows the algorithm used to “encrypt” and “decrypt”. The security of these obfuscation algorithms resides in the secrecy of the key needed to decrypt data. If a key is compromised, the only requirement is to change the value of the key in order to re-protect all data.

The problem is that changing the key is often a delicate process. Key management is exactly the term used for all procedures and techniques used to deal with operations such as key creation, exchange, renewal, and storage. Before discussing key management, let us briefly recall a few basic cryptographic notions.

There are two main cryptographic models:

- In *symmetric cryptography* the same key is used for both encryption and decryption. This family of algorithms, that include the well-known 3-DES and AES, are more efficient but pose the problem of securely sharing the key between sender and recipient.
- In *asymmetric cryptography* mathematically related key pairs are used for encryption and decryption. One of the two keys (the encryption key for communications, the decryption key for signatures) is made public, while the other one is kept secret. Asymmetric protocols, that include the well-known RSA, are more computationally heavy, but they solve all key distribution issues.
- *Hybrid* approaches are often used to combine the strengths of both methods by using asymmetric cryptography to distribute a symmetric key (a session key) used for bulk encryption

Key management is comprised of:

- *Creation of keys*: it requires looking for a cryptographic library that provides for generation of keys using random generation function. That will help users avoiding the management of multiple parties with independent key parts. This way the keys can be generated by the system and humans will never know them.
- *Storage of keys*: it is usually implemented through an additional encryption level: the key used for encrypting data (called a DEK for Data Encryption Key) is encrypted using a specific key for encrypting the storage of the DEK (called a MEK for Master Encryption Key). The DEK and the MEK will need to be stored on separate physical systems so that if one if compromised, the other is not. Even though the MEK is supposed to be stored on a physically-secure system, it is possible to use some further kind of encryption or obfuscation of the MEK, but that is not a requirement from a strict standpoint.
- *Access of keys* for encryption/decryption: it can be dealt with in different manners considering that keys will need to be transmitted across components of the system due to the physical separation of DEK and MEK storage. Either the crypto routines are embedded in the tier using them, or a crypto service is needed which introduces the problem of how data is securely exchanged between application code and crypto services.
- *Key lifetime* (or crypto-period): it is the maximum usage period and lifetime akin to data retention period. Establishing a crypto-period is fundamental because any password, no matter how strong, becomes insecure sooner or later.
- *Key lifecycle*: it consists of (at least) the following states:
    - Current (NIST: Active) – used to encrypt and decrypt data
    - Retired (NIST: Deactivated) – used only to decrypt data
    - Expired (NIST: Compromised) – used only to decrypt data of a compromised key
    - Deleted (NIST: Destroyed) – historical reference to a key that no longer exists

The key state transitions are usually automated in accordance with the key lifetime policy. This is especially true if the data retention period is longer that the combined current and retired key lifetimes as this requires re-encrypting. Managing a compromised key or set of keys is one of the most difficult problems in key management, whose most suitable solution strongly depends on the speficic application settings.

Finally, a fundamental aspect of key management in computer networks concerns whether it is:

- *Centralized*, as happens with Key Distribution Centers

- *Decentralized*, as happens with key update techniques such as key chains (*e.g.*, tokens, otp)
- *Distributed*, as happens when users collaborate to restore secure keys – especially in autonomous, unattended networks

1.7 SUMMARY

Access Control is concerned with limiting the activity of legitimate users and it is critical for security of computer systems. Without the knowledge of the identity of a principal requesting an operation, it is difficult to decide weather the operation should be allowed. In particular, a combination of Identification and Authentication is of primary importance to guarantee correct Authorization and Accountability.

Traditional authentication methods are not suitable for use in computer networks, but proper Administrative, Technical and Physical controls are needed, together with suitable Preventive, Detective and Corrective mechanisms. Technical control must use strong authentication methods that do not disclose password, should consider the possible requirement of remote authentication, and possibly implement biometrics.

Proper access control administration is fundamental. It can be centralized, decentralized, or Single Sign-On. All three have advantages and disadvantages, and the choice depends on the system's requirement and typical usage.

To administer access control, a company needs to choose the most suitable model, mostly among discretionary (DAC), mandatory (MAC) and role-based (RBAC), and to define and implement precise policies, such as Bell-LaPadula or Biba.

In access control, especially if remote, cryptography is of primary importance. To prevent attacks, secure key management is fundamental. In particular, it is important to distinguish between DEKs and MEKs and guarantee proper security of both, and to rely on a KDS if key management is centralized, or on advanced techniques otherwise.