2 Security Goals, Threats and Mechanisms©

The first step in designing or analyzing the security of a system is careful identification of the security requirements and goals, and of the relevant threats and risks. In this chapter, we list common security goals and requirements. For each requirement, we illustrate typical threats that the requirement is addressing, as well as typical, illustrative techniques and mechanisms used to address the requirements (and threats).

2.1 System Integrity – Security against Penetrations

Any secure system must prevent penetrations, i.e. unauthorized access and operations. Indeed, one of the most common attacks on cryptographic systems is direct exposure of the secret cryptographic keys, by system penetration. Penetration attacks involve one or more of the following adversary types:

- Insider attack: abuse by a person with authorized access to the system.
- Hacking: attack via communication links to other systems (e.g. Internet).
- Malicious software: attack on the system by software running on it.

It is important to realize the weaknesses and strength of the two parties – the attacker and the defending security officer. The strength of the attacker is that they choose the time and method of attack, potentially picking on the weakliest protected component – and due to the complexity of real-life systems, it may be impossible or impractical to identify and fix all weaknesses. On the other hand, the security officer has the ultimate control over the system, and therefore once an attack is detected, it is usually relatively easy to block it and prevent the attacker from accessing the system.

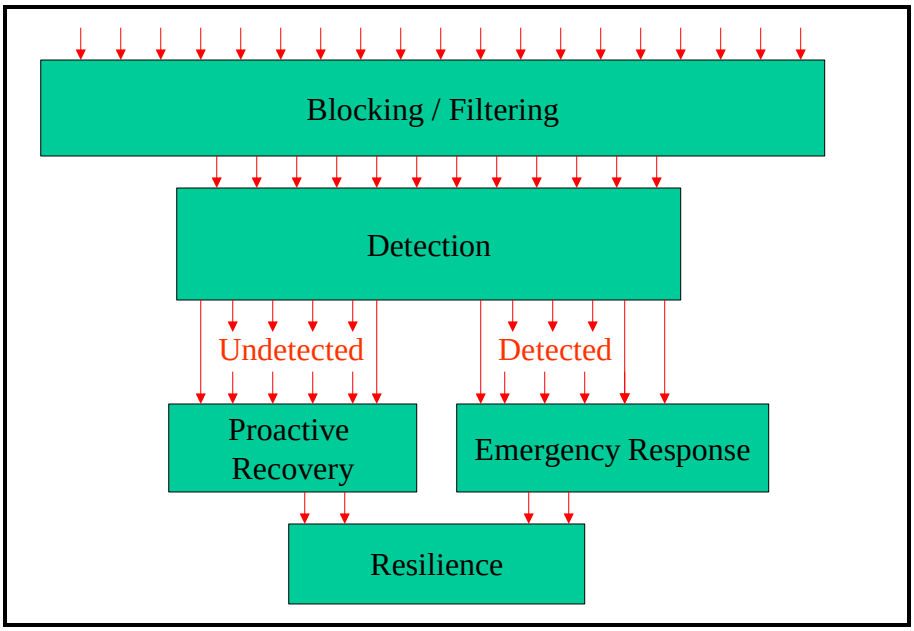

Figure 2.1: Multiple `Lines of Defense`

We conclude that security mechanism can be grouped into multiple `lines of defense`, as illustrated in **Figure 2.1**. These lines of defense are:

1. *Blocking and filtering:* Prevention of penetrations and unauthorized use of a resource. *Firewalls* provide blocking and filtering of interactions with an external networks. Computer security, and secure operating systems, both deal with blocking intrusions to a particular computer, using mechanisms such as user identification (login), access control, execution of (suspect) programs in safe containers (`sandbox`), and other means of isolation and control. Unfortunately, none of the currently popular operating systems is considered secure. In fact, the widely used versions of the Windows operating systems, which are by far most widely used, are extremely insecure. Furthermore, even on operating systems which offer security mechanisms, their proper use is non-trivial and in many cases they are not deployed properly.
2. *Detection:* Detection of activities and situations that (may) indicate penetration attempt. Typical mechanisms are intrusion detection systems and virus detectors. Detection may be triggered by identification of known attack patterns (`attack signature`) or by significant deviation from normal operation statistics.
3. *Emergency Response:* Following detection of penetration attempt, there are variety of tools, services and processes to recover security, identify and close the `holes` (weaknesses) exploited by the adversary, and possibly identify and/or penalize the adversary. These include restoring data and/or programs from backup, using audit trail and monitoring tools, and using services such as of the Computer Emergency Response Team (CERT).
4. *Proactive Recovery:* These are processes and mechanisms that attempt to restore security, but are invoked `proactively`, i.e. not as a result of detection of an ongoing attack. In many cases, these processes will force the adversary to either expose that an attack is taking place, or lose some abilities, for example by

forcing periodical password change. In chapter 7 we describe proactive security achieved by cryptographic protocols, involving cooperation among multiple modules to automatically, periodically restore security.

5. *Resilience:* Since some attacks may escape detection and recovery (or proactive recovery) for substantial time (or forever), systems should be designed, when possible, with automated, secure redundancy mechanisms that provide resiliency to reasonable number of faults. In particular, resilient, redundant design is the main defense against `denial of service` attacks, e.g. using backup and shadow sites. In chapter 7 we describe distributed cryptography, which provide mechanisms for resilient systems using redundancy, e.g. secret sharing.

Unfortunately, adversaries (e.g. hackers) are also aware of the different defense mechanisms, and of the relative advantages of themselves and of the security officers. Namely, serious adversaries will invest much of their efforts in avoiding detection of an attack, to the extent of giving up on acquired (unauthorized) abilities, when keeping these risk exposure of the existence of an attack. In particular, serious adversaries will often combine several attack techniques, using their different characteristics of visibility and ability. In particular, malicious software can be a very effective way for an attacker to gain access to a system, bypassing intrusion detection and prevention mechanisms (e.g. firewalls). By limiting the actions of the malicious software, and making it appear to be an ordinary, useful program, it may escape detection for a long time (therefore, in a serious attack, malicious software will be very cautious about aggressive actions such as modifying other programs, as done by software viruses).

This book is mostly concerned with security in network environments, where cryptography is the main tool. Readers interested in details about computer security, including detection, prevention and recovery from intrusions, penetrations and viruses, are encouraged to read some of the many publications in the areas of computer and network security, e.g. [A01].

2.2 Confidentiality and Privacy

When sensitive information is stored or communicated, there is often a threat of its exposure to an unauthorized party (adversary). An important security goal is to preserve the secrecy of confidential and private[1] information.

2.2.1 Confidential communication

Secure communication often brings to mind the threat of eavesdropping by an adversary, and its prevention via encryption. Figure 2.2 shows a typical scenario, where Alice wishes to send a confidential message to Bob, via a channel that may be subject to eavesdropping by adversary Eve. The use of the names Alice and Bob for the participants in protocol, and of Eve for the Eavesdropper, is common practice and will be used in many examples in this book (along with some other `meaningful` names).

[1] Private information usually refers to information of personal nature, i.e. linked to an individual.

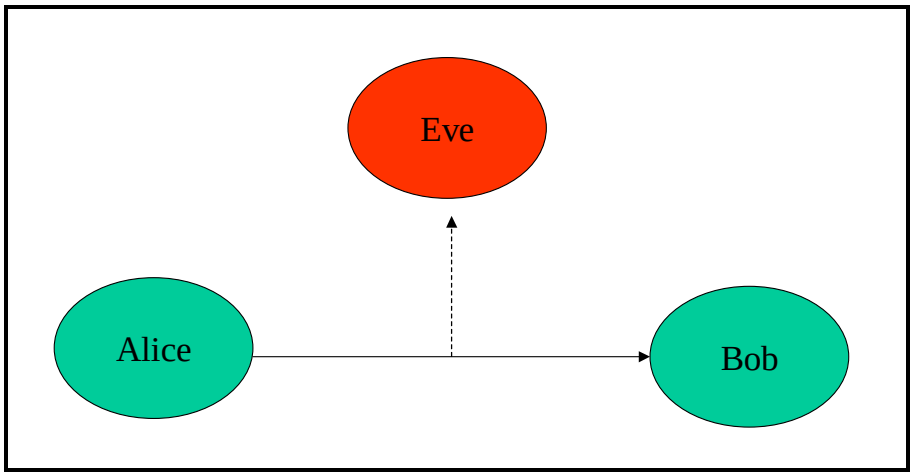

Figure 2.2: Confidentiality requirement and Eavesdropping Threat

The main technique we use to prevent eavesdropping is encryption, which is described in Chapter 4. Namely, Alice sends to Bob an encrypted version of the information, and only Bob has the secret key necessary to decrypt and retrieve the information.

2.2.2 Traffic analysis

Encryption allows hiding of the contents of messages sent over a public network. However, encryption does not hide the length of the messages, and certainly it does not hide the fact the some message was sent from Alice to Bob. In some situations, the knowledge of the existence of transmission (of particular length) between two parties, at a particular time, is sufficient for the adversary. In other situations, information about the patterns of communication of (encrypted) messages among different parties, may provide useful information to an adversary.

Prevention of traffic analysis requires substantial overhead, of adding redundant communication (bits and messages). Therefore, this kind of measure is used only when absolutely needed for a particular application.

2.2.3 Privacy

Companies and organizations often collect large databases of information about individuals, for marketing reasons, to improve and personalize services, or to provide specific services for the individuals, e.g. storage for personal documents and e-mail. Whenever such information is collected, there is the risk of it being abused, e.g. as a result of a penetration of the database. Such abuse may involve serious threats to the individual, such as *identity theft*, where an imposter uses personal data to pretend to be the individual, thereby obtaining credit and other rights, and possibly damaging the individual reputation or assets. Other serious threats include blackmail and discrimination (e.g. on disclosure of insurance risk).

Customers may be concerned about usage of such data, even for seemingly reasonable purposes. There is an ethical issue here, as well as a potential liability risk. In many

countries, there are also legal restrictions on the kind of information kept and its use as well as on the necessary precautions to prevent unauthorized storage and use.

Privacy may be maintained by ensuring that personal information is only kept and used per the user instructions. The main mechanism to achieve this is *secret sharing*, where the private information is split into several fragments, each kept by a separate, independent server. The information can be used only in ways to which all (or most) servers agree.

2.2.4 Location privacy and anonymity

In mobile communication scenarios, the location of the parties may be sensitive information that they parties do not wish to be revealed. A simple solution to provide location privacy is when only the `forwarding agent` knows the current location of each user; more complex protocols try to avoid complete trust in a single `forwarding agent`.

Similarly, parties may be concerned about *anonymity*, i.e. exposure of their identity while traveling and when they communicate or perform transactions (e.g. pay). The simple solution to anonymity is to use a pseudonym instead of the real user identity. A pseudonym may allow an adversary to link between multiple transactions using the same pseudonym, and requires trust in the agents that map from the real name to the pseudonym. In some applications, such as payments, there are alternative solutions which ensure anonymity without trusting any server or agent.

2.3 Unauthorized and Disputed Operations Threats

Whenever two parties communicate, there is the threat of miscommunication – the message received is somewhat different from the intention of the originator. In the context of a malicious adversary, this threat is more severe: the adversary tries intentionally to cause some disallowed, *unauthorized operation*. To achieve this, the adversary may try to *impersonate* the originator, and assume its identity in the eyes of the receiver, often causing the receiver to accept a message that the originator never sent. Or, the adversary may try to *modify* a message on transit, causing the recipient to receive a different message from the one sent, or tamper with the sequence of messages, e.g. reorder or duplicate messages to cause an unauthorized operation.

A related threat is that of a *disputed operation*. In this case, an (authorized) operation was performed, e.g. Bob carries out some request made (supposedly) by Alice. However, later the requestor (Alice) denies having made the request, raising a dispute. Disputes are usually the cause of some malicious attack, by one of the parties (Alice or Bob) or by an attacker impersonating as Alice.

We now elaborate a bit on the unauthorized operation and disputed operation threats, and discuss cryptographic mechanisms that can help address them.

2.3.1 Entity Authentication and Identification

We first discuss the threat of impersonation, namely of the adversary assuming the identity of one party, say Alice, when interacting with another party, say Bob. Intuitively,

the goal is to ensure Bob that the other party is really Alice. However, this problem has several important variants.

First, there is a big difference between the cases where Alice is a person and where Alice is a computer. When Alice is a person, the problem is usually called *identification*, and there are three ways for Alice to prove her identity:

- By *something she knows*, i.e. proving she *knows* some *secret*, e.g. a password. This is *secret-based identification*.
- By *something she has*, e.g. a smartcard. This is *device-based identification*.
- By *something she is*, i.e. demonstrating some *personal, physical property*, such as fingerprint or voice. This is *biometrics-based identification.*

In *biometrics-based identification*, as in **Error! Reference source not found.**, there must be some trusted measuring device that measures the biometrics. Bob must have secured communication with the biometrics-measuring device. In fact, a very good way to use biometrics is as a mechanism to identify the owner of a mobile phone or computer, in which case the measuring device will be integrated with the mobile phone or computer (acting as `Bob`). If Bob and the device communicate via an insecure channel, we may use a cryptographic key built into the device to protect the communication. The biometrics data itself, allowing identification of Alice, may be accessed by the measuring device itself, but more usually is kept by Bob. Security relies on the integrity of this database, on the integrity of the secure measuring device, on the security of communication, and on the inability of Eve to provide indistinguishable biometrics from Alice's.

Biometrics-based identification is often expensive, difficult to deploy and inconvenient to use. In particular, being based on measurement of unique physical properties, the measurement is clearly subject to some errors of two kinds: false warnings and undetected fraud. In addition, biometrics, by their very nature, may be considered as exposing the privacy of the individual, and storing and transmitting biometrics data may cause resistance, introduce liability and be subject to legal restrictions (see Subsection 2.2.3 above).

However, some biometric mechanisms can be very conveniently integrated with appropriate devices, e.g. voice recognition integrated into a phone. Furthermore, when used to authenticate the user to her device, several of the risks above may become irrelevant.

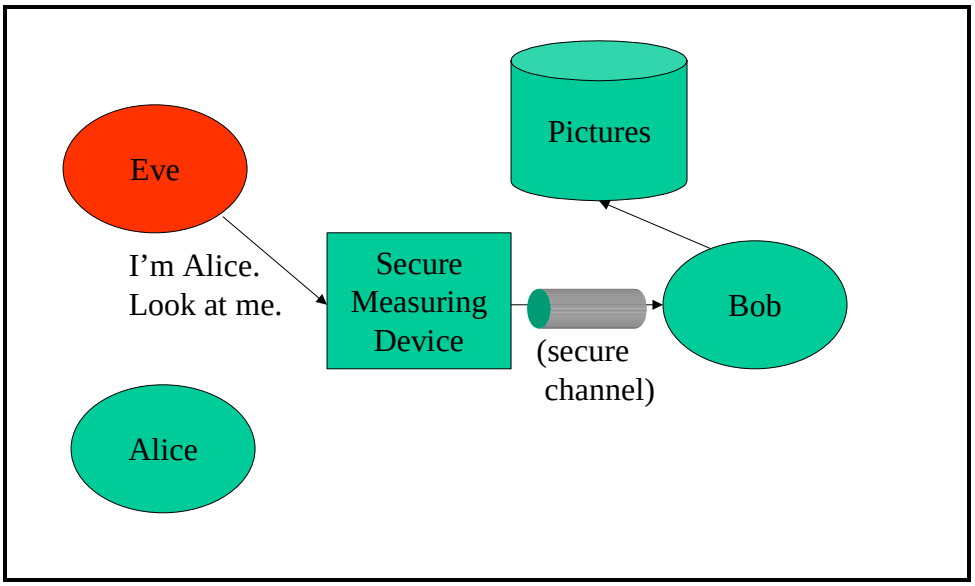

Figure 2.3: Biometrics-based Identification

Secret-based identification, as in Figure 2.4, must also involve a secure device for entering the password (or other secret, e.g. PIN – Personal Identification Number). The device has to ensure proper processing, e.g. not expose the password,, and some secure way to indicate to then user when he can enter his password safely (e.g. a special button). This scheme is also used often with weakly secure devices (e.g. a PC running Windows), but then it may be subject to spoofing attacks, which cause the user to enter his password into the malicious software. Some solutions also use a secure channel using a secret key stored permanently by the secure terminal. However, most solutions try to avoid this assumption, by using the password (or other user-supplied secret) to secure the communication.

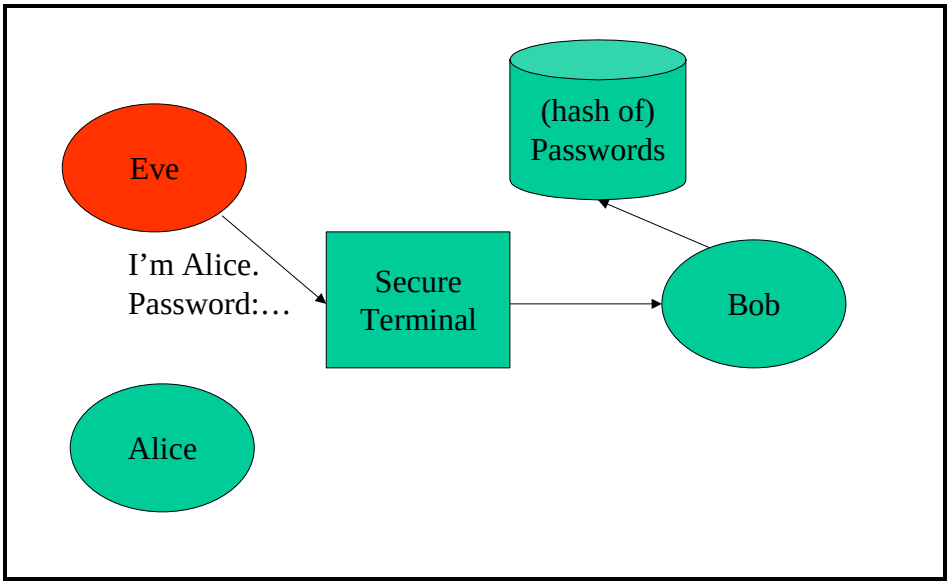

Figure 2.4: Secret-based Identification

There are different protocols for checking passwords, from simple ones where passwords are sent on the clear, to complex protocols that deal with threats such as:

- Eavesdropping on the communication between the secure terminal and Bob.
- Exposure of the file containing password information kept by Bob. Solutions usually involve keeping only the hash of the passwords.
- Password guessing (*dictionary*) attacks, exploiting the fact that when people are allowed to pick passwords, they tend to pick weak passwords. In an *offline dictionary attack*, the attacker captures some message and uses it to test password guesses. Solutions include mechanisms to prevent or discourage the use of weak passwords, and mechanism to prevent offline dictionary attacks (force the attacker to actually interact with Bob for each guess).

The biggest problem with secret based identification is the fact that most users do not properly use passwords (or other secrets), since proper use is quite inconvenient. Some of the wrong things people do with passwords include:

- Choosing weak passwords, i.e. passwords from a relatively small subset of all possible combinations, which may allow the adversary to guess the password. People choose weak passwords to make them easier to remember (and sometimes also easier to type). Adversaries exploit this by dictionary attacks (as discussed above), where they guess passwords out of a dictionary of likely candidates. The dictionary will include common words and names (e.g. of people and of fiction), and potentially also common combinations of names and identifiers associated with the individual whose password is being guessed (e.g. spouse or children names, car license plate, etc.).
- Reusing passwords, when passwords, or minor variations of them, are used for authenticating for multiple parties. This allows exposure of the password from one party to be used to penetrate the user's account with another party. Furthermore, when the passwords used with different parties have different rules, this may help the attacker to cut down on the number of guesses. For example, a user
- Storing passwords (in insufficiently secure location). Since people need to use passwords for so many systems, they tend to write them down (on paper, in a file, etc.), as well as often allow their automated storage (e.g. by a browser). When such storage can be accessed (e.g. by a malicious program such as a virus), the passwords are exposed.

The tendencies of people to pick weak passwords, to reuse passwords and to store them, have all increased with the growing usage of passwords and other secret identifiers by many devices, computers and services. Many systems also require users to frequently change their passwords, to make it harder for an attacker; yet this adds additional burden on users (who often respond by using simple rules for slightly changing their passwords, again potentially causing them to become weaker). All of this motivates the use of other forms of identification, such as biometrics and secure personal devices.

Device-based identification, as in Figure 2.5, involves a secure, personal identification device that belongs to and carried by each party, say Alice. The device usually contains

some secret key, and the other party, say Bob, has a corresponding key and/or other information, used to securely identify Alice's device.

By virtue of the key and cryptographic capabilities, device based identification is (if well implemented) secure against attack on the communication between Alice and Bob. They can also be multipurpose, and identify the user to multiple servers, which can reduce costs and improve convenience.

There are several hurdles for device-based identification:

- Device cost is sometimes a prohibitive factor.
- The user often has to manually copy information from the device to the terminal (and sometimes vice verse as well), due to lack of universal communication interfaces. For example, the device may be only a tiny screen that displays identification codes that change each second, and Alice copies the code into the terminal.
- Devices may be lost or stolen. Preferably the device will contain additional identification mechanisms, secret or biometrics based, to ensure that only the owner can identify with it.

We anticipate that in the future device-based identification will become pervasive, as an added functionality to personal devices such as mobile phones. The identification service will be applicable to arbitrary application, providing substantial convenience to users. The extra cost will be negligible, and the device will have built-in mechanisms to identify the user (maybe using voice recognition) and to communicate with the terminal.

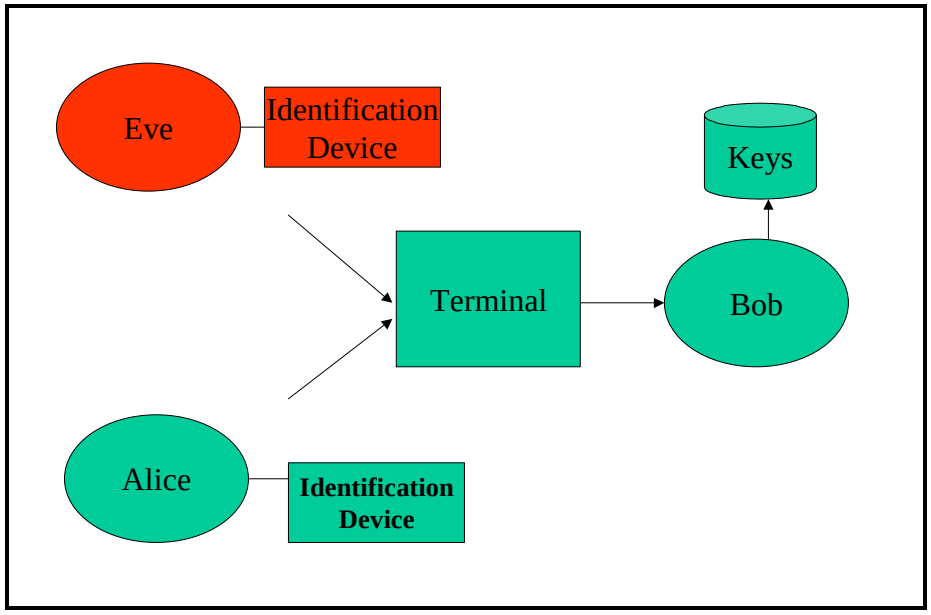

Figure 2.5: Device-based Identification

With such stronger identification devices, it may be possible to communicate directly to Bob. In this case, the terminal becomes unnecessary, and there is no need to distinguish

between Alice and its device as all communication is via the device. This scenario is identical to the entity authentication scenario, which we show in Figure 2.6.

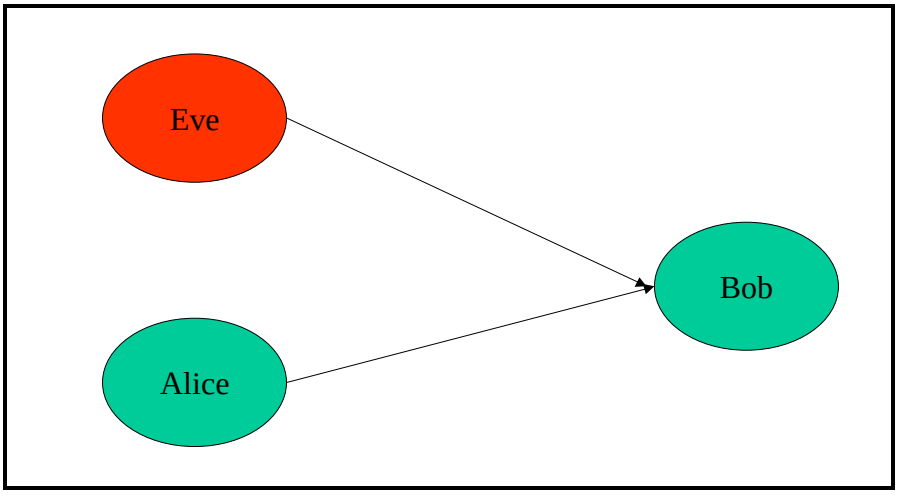

Figure 2.6: Entity Authentication

In entity authentication, Alice and Bob are computers, which authenticate each other (*mutual authentication*) or only one of them (typically in client-server environments, with *client-only authentication* or *server-only authentication*). Authentication may be of the exact identity of the peer, or of some general properties and *credentials*, e.g. being an arbitrary member of some group. The basic threat here is *impersonation*, where an adversary, Eve, tries to present incorrect identity or credentials. The impersonation threat can be achieved by one of the following attacks:

- *Replay attack*, where the adversary, Eve, uses messages from past authentications to impersonate.
- *Interleaving attack*, where Alice and Bob may run multiple authentication sessions concurrently, and Eve intercepts messages from some of them to impersonate on others.
- *Man in the middle attack*, as in Figure 2.7, where Eve simply forwards all messages from Alice to Bob and vice verse, until one of them (say Bob) grants Eve some access or privilege thinking it is Alice. Forwarding messages is often possible, when communicating over open networks, therefore the right way to cope with this attack is to authenticate all communication between the parties (requests, responses etc.), not just some initial exchange. This makes the forwarding (man in the middle) attack meaningless. Initially, or possibly periodically, we establish the identity and/or credentials of the peer, as well as keys for authenticating communication between them.

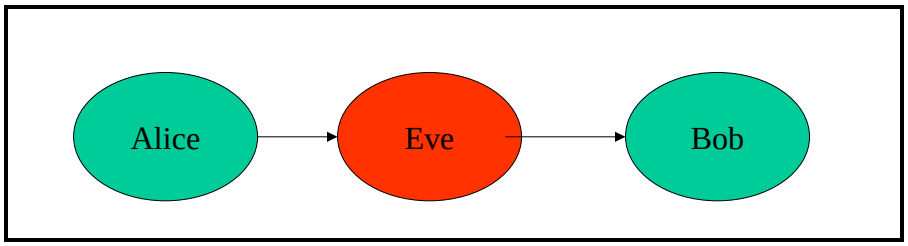

Figure 2.7: Man in the Middle Attack

2.3.2 Authorization and Correctness

In the design of a secure system, we need to define clearly the set of allowable states of the system. This set is different for every application, and may include properties such as fairness and fraud prevention. In general we refer to this set of allowable states (and/or disallowed states) as the *correctness* definition.

While the general correctness definition is very general and application specific, *authorization* is often an important part of it. Almost any system has a set of allowable, authorized operations for each participant, and possibly also a set of explicitly disallowed operations. *Unauthorized operations* are a basic threat, and a common goal for adversaries, which often can gain directly from carrying an unauthorized operation. For example, an accountholder is (normally) not authorized to increase the account balance, but has a direct gain from finding a way to do so.

Entity authentication, discussed above, is a basic requirement to prevent unauthorized operations, by authenticating their identity and/or their credentials. *Authorization* ensures that the entity can only perform actions according to its rights, as defined by its identity and/or credentials. In the case where the operations are well-defined forms of access to specific resources, e.g. files on a computer, the authorization problem is often referred to as *Access Control*.

The set of authorized (and/or unauthorized) operations is very application dependent. Often, the application definition includes definition of the authorized operations to different entities (based on identity or credentials). Examples of such `static authorization` applications include:

- Payment mechanisms, where only the account holder is authorized to pay, and even the account holder cannot pay more than the funds in the account, etc.
- Voting, where each participant can vote only once.
- System management, where only the operator can shut down the system.
- Distribution of copyrighted content, where the consumer is not allowed to copy or modify the content.

In other cases, the mapping of rights based on identities and credentials can be defined as a *dynamic policy*, defined by decisions of entities that have management or ownership rights, and possibly changing over time. Examples of such `dynamic authorization policy` applications include:

- Access control, where the owner of a resource (or file) can grant or deny access to it.

- Delegation mechanisms, where an entity can delegate some rights to another.

2.3.3 Message Authentication and Integrity

Message authentication allows the receiver of a message to verify who sent it, and that the message was not originated, or changed (modified) by some intermediate entity. To achieve this goal, we usually send the message together with a verification field computed by Alice and used by Bob to confirm that the message is authentic (from Alice) and unmodified. This verification field can be a digital signature generated by Alice (using a secret key) and verified by Bob (using a public key). However, digital signatures are computationally intensive, therefore we usually use instead *Message Authentication Codes* (*MAC*, also called Message Digest and Message Integrity Code or MIC), computed and validated using a secret key shared between Alice and Bob.

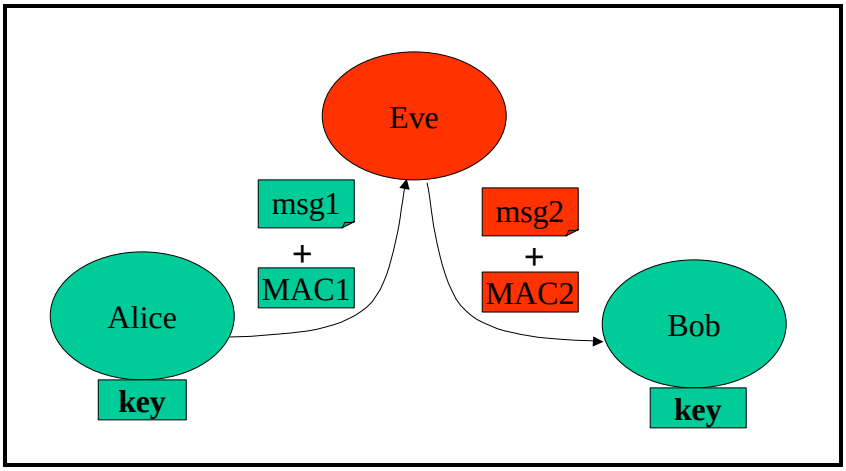

Figure 2.8: Message Authentication - Prevent Message Modification/Injection Threat

The threat this defends against is message modification and/or injection, shown in Figure 2.8. The adversary generates some message, and sends it to one of the parties, e.g., trying to convince it to deposit some amount of money in his account. The method of generation can be, e.g., by changing the amount or account number in a legal deposit message.

2.3.4 Ensuring Reliable Communication Sequence

When parties communicate over a network, they usually send and receive multiple messages, relying on network protocols to ensure that messages are received exactly as sent – with no replay (double transmission), omissions or reordering – and within bounded delay. An important requirement of many secure systems is to ensure reliable communication – in spite of adversary's attempts to introduce errors. The different kinds of errors that an adversary may try to introduce are illustrated in Figure 2.9 to Figure 2.11.

We now discuss each of these errors briefly:

- *Message delay attacks* (Figure 2.9): the adversary tries to deliver a message after delaying it intentionally, for much longer than any possible communication delay. This is solved by attaching a *timestamp* (clock value) to the messages,

when synchronized clocks are available, or by requiring periodical handshake exchange, regardless of the need for sending new message.

- *Message replay (duplication) attack (*Figure 2.10): attacker sends two or more copies of a message that was sent once.
- *Message reordering attack* (Figure 2.11): the adversary tries to change the order of received messages.
- *Message omissions attack* (Figure 2.12): the adversary tries to remove one or more messages from the stream transmitted, without awareness of sender and receiver.

There are three basic approaches to ensuring reliable communication and preventing replay, reordering and omissions attacks:

- Including a random challenge with each request, to be copied to the response. This is the preferable solution, except where the additional communication of challenge from recipient to sender is impossible or difficult.
- Include sequence number with each message. This requires reliable storage of the counter, never to be reset, which is sometimes difficult.
- Include time with each message. This is much like sending a counter, but may be easier to implement in some systems. Additional protection is necessary against omissions, such as including the time of the previous message as well.

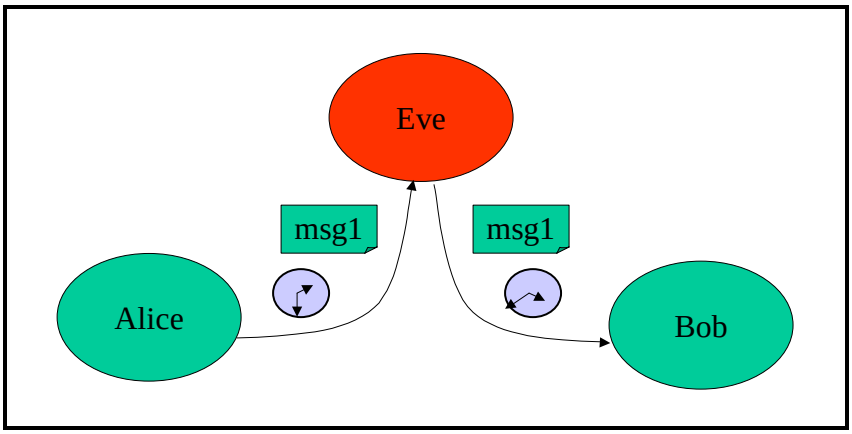

Figure 2.9: Message Delay Attack

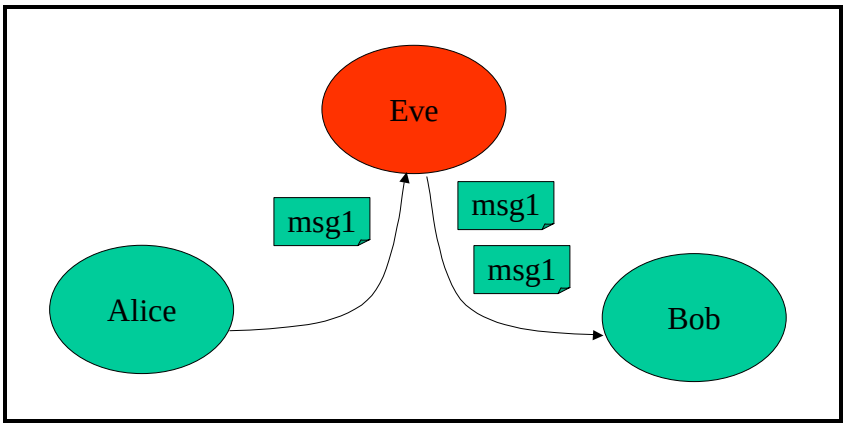

Figure 2.10: Message Replay (Duplication) Attack

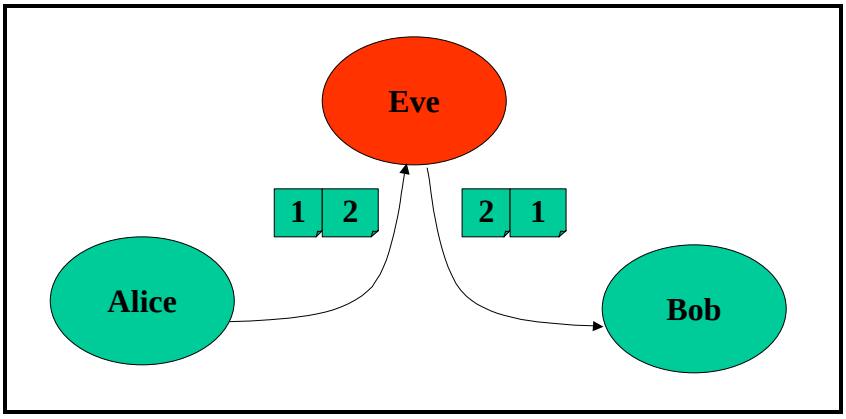

Figure 2.11: Reordering Attack

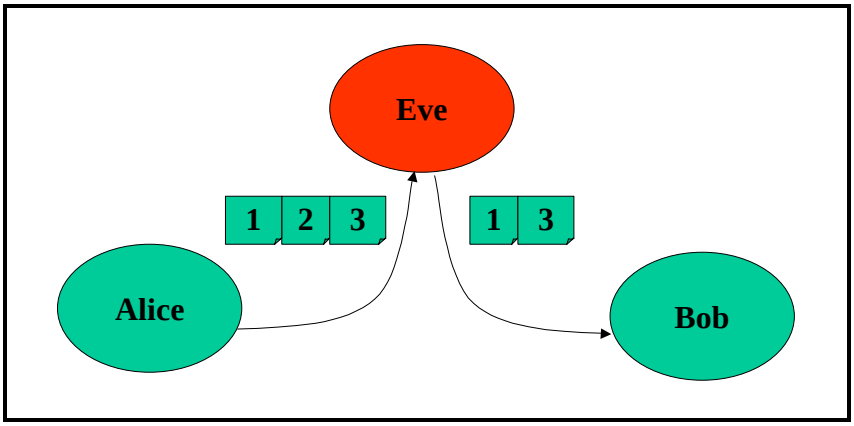

Figure 2.12: Omission Failure / Attack

2.3.5 Non-Repudiation

Non-repudiation is binding of an entity to an action or statement it made, in particular, being the origin or the destination of a message (sent at a given time). This prevents the entity from disclaiming the action or statement.

The main cryptographic tool for non-repudiation is (public key) digital signature. The signer computes the digital signature over the statement using a secret key, and anybody can validate it using a public key.

2.3.5.1 Non-repudiation of Origin

Non-repudiation of origin binds an originator of a message, so that a recipient can prove later, to a third party, that the originator sent the statement. This prevents an adversary from sending a message, e.g. a payment order to some other party, and then he tries to gain profit by claiming that he never sent this message. With non-repudiation of origin, the recipient has a `proof` of the identity of the originator, which it can show to a third party (e.g. judge). This `proof` is usually a digital signature.

2.3.5.2 Receipts - Non-repudiation of Delivery and Timestamp

Many electronic commerce applications require signed and timed receipts for messages (e.g., for an order sent to a stock broker). Such receipts prove that the message was properly delivered (by a given time). This allows later resolution of disputes, by the parties or by a trusted third party (judge). A trusted party, often called *notary,* generates the receipts and timestamps. The parties must agree in advance to accept receipts originated (signed) by the notary as sufficient proof of delivery.

2.4 Availability

It is often easier to break something than to build it or to mend it. *Denial-of-service attacks* attempt to disrupt services. The system designer attempts to ensure availability, make it harder to disrupt availability, or at least detect when an attack prevents availability.

2.4.1 Ensuring Availability – Preventing Clogging Attack

In a clogging attack, the attacker is an external entity, whose goal (for financial, ideological or other motives) is to cause a failure or reduced performance of the server. The attacker tries to achieve this by generating a huge number of fake messages, and sending them to the server. These messages can be constructed as random messages or as copies of old messages with or without changes. If the system provides authentication, then the server will recognize the invalidity of these messages. However, when the resources required for invalidating such message are larger than those required for sending it, the server might fail handling all these messages, as well as valid messages from real users. The result of this scenario might be a complete failure of the server.

Clogging attacks on most servers cannot be completely eliminated, as long as any client can send messages to the server. In this case, the attacker can mount a *distributed denial of service attack (DDOS),* as we show in Figure 2.13 below, where the attack is launched simultaneously from many computers. The attackers often use many computers by first

breaking into them, by an automated penetration mechanism, e.g. malicious software such as a virus, or automated hacking program. It is unfortunate that current popular operating systems are extremely vulnerable to penetrations and malicious software, making a huge number of computers on the network a potential resource (and hence a target) for hackers.

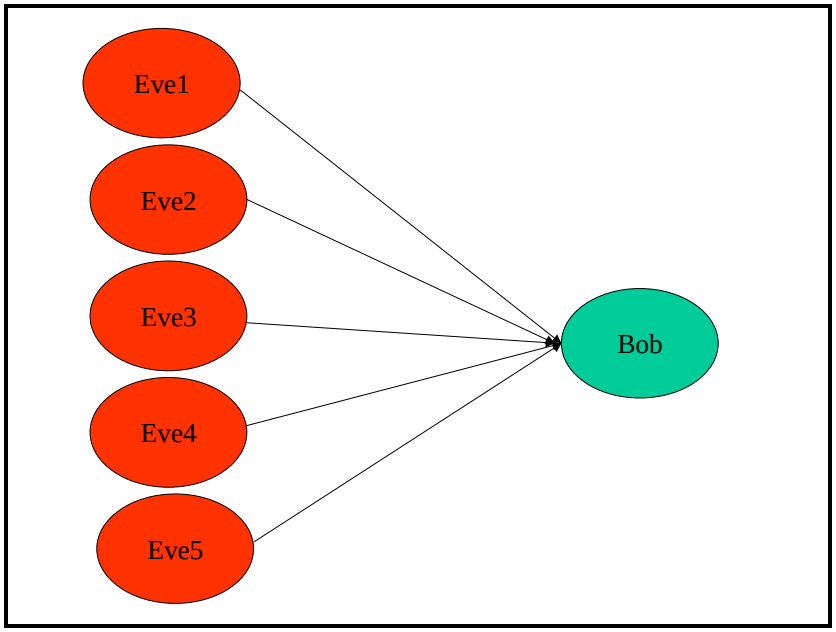

Figure 2.13: Distributed Denial of Service (DDoS) Attack

A huge sequence of invalid messages will always cause delays in handling the legitimate requests. However, it is possible to minimize their effect as much as possible, e.g., by efficient authentication of messages before any heavy processing takes place or any state is allocated. Most means for prevention of clogging attacks try to maximize the ratio between the amount of work of the attacker and the amount of work that is caused to the system.

2.4.2 Detection of Disconnection

Many e-commerce applications rely on the availability of communication and services. An obvious threat is for an attacker to disconnect the communication, or take down some of the participants, without the others being aware of it. We often address this threat by sending `keep alive` messages periodically between the parties, to detect if the communication is down. The `keep alive` messages should be authenticated to prevent the attacker from simulating them.

2.5 Usability and Unambiguity

Many security mechanisms offer sophisticated, advanced features, but unfortunately are not widely used or often used incorrectly. This is often due to the conceptual complexity of security and especially cryptography, and the fact that lack of security is evident only when attacked, not in normal operation. This also implies that when security tools are used, correctly or incorrectly, users often have unrealistic expectations about the security

they provide, leading to a false sense of security. These problems present major threats to security:

- *Excessive usage or deployment complexity,* causing the systems to be unused or used incorrectly, and
- *False sense of security,* due to vague or excessive claims and/or to poor system management.
- *Inadvertent breach of security,* where an attacker exploits the user's lack of understanding of security mechanisms, and the vague or insufficient warnings provided to the user, to cause the user to inadvertently assist the attack. Some examples are:
  - *E-mail vandals:* malicious programs that are sent as (attached) part of e-mail messages. Users are encouraged (by the text in the message) to execute these programs, thereby allowing them to perform unauthorized operations (including, usually, forwarding similar messages containing the vandal to other users).
  - *Active malicious remote code:* malicious programs that are downloaded e.g. by the browser. Browsers prompt the user before allowing such code to execute, but users often do not understand the warning well.

2.6 Exercises

1. A bank offers customers access to their account via Internet, phone and ATM. The bank official decided it would be easier for the customers to use the same PIN for all three purposes. For highly sensitive operations such as money transfer, a 16-digit PIN code must be used; for simple queries, such as account balance, the first four digits of the PIN code are sufficient, and for some medium-sensitivity operations such as investing, the first eight digits of the PIN are sufficient (or only four in the ATM). Criticize.
2. Many banks offer home banking services via the web, protected by a password. The user is responsible for any transaction authorized using the password associated with the account. Describe at least three major problems with this.
3. A firm proposes to improve home banking security by installing a fingerprint reader connected to the user's computer, and sending the fingerprint instead of a password. Criticize.
4. Which security threats and requirements are relevant, and which are not, to the following cases:
   1. Sending a complaint on a peer employee for sexual harassment.
   2. Sending an audit report to the CEO.
   3. Receiving bids to an online auction of government property.
   4. Sending a message describing the status of army units, by a spy, to his home country, over the Internet.