Introduction to Security - Notes

Stelios Gisdakis

November 6, 2014

Contents

1 Introduction to Security

1.1 Why Security

The communication could be attacked by a malicious adversary. There are two different kinds of attacks: passive and active. In the first kind, the attacker simply tries to reach his goal without affects the communication. As for the second kind, the attacker actively tries to break the system.

1.2 Eavesdrop

Example of basic threat: eavesdropping. The attacker tries to acquire the data by for example listening the channel. The data could be a user password for future use or confidential information. Page on OWASP: https://www.owasp.org/index.php/Network_Eavesdropping.

1.3 Disruption

Example of basic threat: disruption. The attacker actively tries to prevent the two ends from communicating.

1.4 Forge

Example of basic threat: forge. The attacker in this case creates usually new messages, valid or not, to compromise the communication. An example could be the IP spoofing, where the attacker changes the value of his source address in the IP header to usually hide himself. Another example could be the ARP spoofing, when a malicious station tries to associate its Medium Access Control (MAC) address to another IP address, usually for taking control of the communication.

1.5 Combining: Replay

Combination of eavesdropping and forging to replay legit packets. The attacker has no need to actually know the contents of the messages and they could still be considered valid.

1.6 Combining: Man-in-the-Middle

Combination of eavesdropping, disruption and forging for taking control of a connection. The attacker usually establishes a connection between both ends and he tries in each connection to impersonate the other connection's end.

2 Security Goals

Formal definition of the security goals, inspired by [1].

- **Confidentiality**: No unauthorized entity can access the content of the communications.
- **Integrity**: No unauthorized entity can modify content of the communications.
- **Availability**: The services offered by the system should be accessible to authorized users.

2.1 Security Goals: requirements

Basic requirements for reaching the security goals.

- **Authentication:**: An entity claiming an identity must be *authenticated*, i.e., prove its identity.
- **Authorization**: What actions (read, write, execute) can an authenticated entity perform over the resources of the system ?
- **Accountability**: The process of monitoring what entities do over the resources of the system (example: user X opened file F)

2.2 Security Goals: requirements (cont'd)

Consequence of enforcing the security goals: non-repudiation.

- **Non-Repudiation:**: No entity can deny having performed an action or having generated a message.

2.3 Authentication

Definition of authentication. Proof of identity.

To prove our identity we rely on the following:

- **Something we know**: A secret piece of information that a user knows and uses to get authenticated to a server (example: password).
- **Something we have**: A secret piece of information that the user has and can use it to get authenticated to a server (example: credit card, digital certificates, ...)
- **Something we are**: Traits of our body/behavior that can uniquely identify us (iris, fingerprint, typing patterns, ...)
- **Where we are**: Our location (physical or not). For example when we are connected to the university network we can access the resources of the university (printers, e-library, ...)

2.3.1 Authentication (cont'd)

Enhance authentication by combining more requirements: strong authentication.

Example: Use password + a credit card reader to perform on-line bank transactions. It is harder for an attacker to get both!

2.3.2 Authentication: Challenge Response

Authentication as a challenge that can be solved only by valid users.

Simple password authentication (when the password is transmitted in clear-text over the Internet) are not secure. To get authenticated without transmitting the password we can use a family of authentication protocols called *challenge-response* protocols. In such protocols, the authenticator issues a challenge to the entity that wishes to be authenticated. This entity receives the challenge and computes a response by using the secret piece of information and sends it back to the authenticator. The authenticator computes the received response with the desired resource and grants/denies access accordingly. One way to implement challenge-response protocols is with the use of hash functions and encryption algorithms (discussed later).

2.4 Access Control

Combining authorization, authentication and auditing for enforcing the security goals. The principle of least privilege requires that each user *must be able to access only the information and resources that are necessary for its legitimate purpose*[1].

The access control can be based on list (**Access Control List (ACL)**) which define which entities (i.e., users) have access (read/write/execute) over the a resource of the system example: <file1: user1(read/write),user2(read)>.

[1] http://en.wikipedia.org/wiki/Principle_of_least_privilege

Another approach is to use **capabilities**. This way we define what all the access rights of an entity example: <user1: file1(read/write),file2(read)>

3 Cryptographic Tools

Introduction to Cryptography. Usually it is difficult (if not even impossible) to prove that a system is secure by design, but it can be proved an attacker could not break it in useful time.

3.1 Hash Function

Hash Functions are efficient **one-way** functions, i.e., it is easy to compute the output of an input but computationally very hard to find the input which corresponds to a given output. Hash functions have **fixed output size**. In principle they take long inputs and transform them into smaller pieces of data (here we talked about the pigeonhole problem in order to explain attacks against hash functions).

3.1.0.1 Hash Functions: properties

Properties of hash functions (represented as a function f) are:
Pre-image resistance: Given an output y of a hash-function it is hard to find the input x so that $f(x) = y$.

3.1.0.2 Hash Functions: properties (cont'd)

Second pre-image resistance: Given an output x and y so that $f(x) = y$ it is hard to find a $x^{'}$ for which it holds $f(x) = f(x^{'}) = y$

3.1.0.3 Hash Functions: properties (cont'd)

Collision resistance: It is hard to find a pair x and $x^{'}$ so that $f(x) = f(x')$

3.1.0.4 Hash Functions: properties (cont'd)

Avalanche effect: desirable properties for cryptographic hash function. Without this property the attacker could easily make prediction about the input, leading to a break of the algorithm.

3.1.1 Hash Functions: use cases

Possible ways to employ hash function. Hash tables are used for example for fast indexing, since the data could be accessed directly by only knowing the hash. On the other hand, hash chains could be used for authentication as one-time passwords: the bank generate the chain and provides it to the user, that will use it in a reverse manner since from $H^n(W)$ is hard to find $H^{n-1}(W)$. Hash trees are used for example to validate chunks of a file in Bittorrent (We will discuss hash-trees when we talk about routing protocols).

3.1.2 Hash Functions: use cases (cont'd)

The main use of hash functions is to ensure the integrity of the communications. Imagine that we transmit a message over the network and that due to communication errors, the message is modified. How can we detect its modifications? If we include a checksum of the original message (i.e., hash the message and include it along with it) the recipient will get the message, compute the checksum and compare it to the one it was included. If the two messages match then everything was OK. Otherwise it was modified.

Important: Simply hashing a message does not prevent malicious modifications. To do this we need a key (discussed later).

Another way to use hash functions is to implement challenge response authentication protocols. The response computed by the entity that wants to be authenticated is the hash of the password along with nonces (random values).

3.1.3 Hash Functions: model

A possible schema for building a collision-resistant cryptographic hash function form a fixed block hash function is the Merkle–Damgård construction[1]. The message m of length t is split in multiple blocks m_i of length L. Each block m_i is given as an input of the hash function together with the $H(m_{i-1})$. The first block, instead, goes with a known Initialization Vector (IV). The message is also padded with 0 to be aligned.

3.1.4 Hash Functions: implementations

Some diagrams for common algorithms: Message Digest v5 (MD5) and Secure Hash Algorithm (SHA)-1. A, B, C, D, and E are 32-bit words. F is a nonlinear function. K_i is the round constant of round i. M_i and W_i denote a 32-bit block of the message input, and $<<<_n$ is the left bit rotation of n places.

3.2 Encryption

Definition of encryption: making the message *plaintext* unreadable to everyone except the ones with the right knowledge (key). Kerckhoff's principle (Claude Shannon reformulation), originally it was[2]:

[1] http://en.wikipedia.org/wiki/Merkle%E2%80%93Damg%C3%A5rd_construction

[2] http://en.wikipedia.org/wiki/Kerckhoffs's_principle

1. The system must be practically, if not mathematically, indecipherable;
2. It must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience;
3. Its key must be communicable and retainable without the help of written notes, and changeable or modifiable at the will of the correspondents;
4. It must be applicable to telegraphic correspondence;
5. It must be portable, and its usage and function must not require the concourse of several people;
6. Finally, it is necessary, given the circumstances that command its application, that the system be easy to use, requiring neither mental strain nor the knowledge of a long series of rules to observe.

3.2.1 Symmetric Key

Symmetric Key encryption, advantages and disadvantages.

3.2.1.1 Symmetric Key (cont'd)

. Different schemes of encryption: stream and block ciphers. The stream ciphers come from the "perfect" cipher One-Time Pad (OTP). The OTP is impractical since the key should be long as the message and it can never be used twice, since $c_1 = m_1 \oplus k$, $c_2 = m_2 \oplus k$, $c_1 \oplus c_2 = m_1 \oplus m_2$ and so the attacker could retrieve part of the messages. The stream ciphers try to generate an OTP from a smaller key. The block ciphers have various operational modes, depending on how the block input and output are chained with the next ones[1].

3.2.1.2 Examples: WiFi Security

. Wired Equivalent Privacy (WEP) uses a stream cipher. The same traffic key must never be used twice as previously said, but the 24-bit IV is not long enough to prevent any repetition in a busy network. Thus, it is relatively simple to break a WEP-protected WLAN.

3.2.1.3 Examples: WiFi Security (cont'd)

. Diagram for Wi-Fi Protected Access (WPA) and Message Integrity Code (MIC) employed in the WiFi. The first is used to generate a new 128-bit long *per*-packet key, thus preventing the types of attacks used for compromising the

[1] http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

WEP. In WPA version 2 (WPA2) the Temporal Key Integrity Protocol (TKIP), that uses Rivest Cipher 4 (RC4), has been replaced with Counter Mode with Cipher-Block Chaining (CBC) MIC Protocol (CCMP) that use the Advanced Encryption Standard (AES)[1]. The MIC is used for assuring integrity and authentication of the message, to avoid the attacker alters o resends frames.

Note: We have a special module on wireless security so do not feel scared if you do not understand these at this point.

3.3 Asymmetric Key

Description of the asymmetric keys encryption scheme, advantages and disadvantages.

3.3.1 Asymmetric Key (cont'd)

Computational security of asymmetric key encryption: there are still no efficient (polynomial time) algorithm to solve a NP problem. Two asymmetric key encryption algorithms are: RSA[2] and ElGamal[3].

Example:

- Choose $p = 3$ and $q = 11$
- Compute $n = p * q = 3 * 11 = 33$
- Compute $\phi(n) = (p - 1) * (q - 1) = 2 * 10 = 20$
- Choose e such that $1 < e < \phi(n)$ and e and n are co-prime. Let $e = 7$
- Compute a value for d such that $(d * e)\%\phi(n) = 1$. One solution is $d = 3[(3 * 7)mod20 = 1]$
- Public key is $(e, n) = (7, 33)$
- Private key is $(d, n) = (3, 33)$
- The encryption of $m = 2$ is $c = 27\%33 = 29$
- The decryption of $c = 29$ is $m = 293\%33 = 2$

3.4 Message Authentication

Descriptions and diagrams of Hash-based Message Authentication Code (HMAC) and CBC Message Authentication Code (CBC-MAC). It is possible to authenticate a message by creating an hash of a message with a symmetric shared key. It is also possible to consequently encrypt the message with a block

[1] http://en.wikipedia.org/wiki/CCMP
[2] http://en.wikipedia.org/wiki/RSA_(algorithm)
[3] http://en.wikipedia.org/wiki/ElGamal_encryption

cipher as shown in the figure. An IV concatenated to all the parts P_i of the message are encrypted with the encryption algorithm E_k used with key k.

3.5 Message Authentication (cont'd)

As we saw before, an asymmetric cryptographic scheme is based on two keys; a public pub_{key} and a private key pr_{key}. Public keys are stored inside **digital certificates** (discussed later). To generate a digital signature of a message m we hash the message and the output is encrypted with the private key pr_{key} of the sender: $signature = Enc(h(m))_{pr_{key}}$

3.5.1 Digital Signature: properties 1/3

Description of Publicly verifiable property. Anyone that knows the corresponding public key pub_{key} can verify the signature by decrypting with it: $Dec(Enc(h(m))_{pr_{key}})_{pub_{key}}$

3.5.2 Digital Signature: properties 2/3

Description of transferable property. When we transmit the message we also transmit the signature: $< m, signature >$

3.5.3 Digital Signature: properties 3/3

Description of non-repudiation property. Only the holder of the private key can generate the signature of a message (and thus cannot deny it).

3.5.4 Public Key Infrastructure (PKI)

Let us talk about certificates now. A digital certificate binds a public key with the identity of the holder of the corresponding private key. Imagine a certificate as the following structure $< serial, Identity, pub_{key}, \sigma_{CA}, validity >$. With σ_{CA} we denote a digital signature of a certification authority (we saw before how digital signatures are generated).

A certification authority is a trusted third party that issues certificates. There might be multiple certification authorities that issue certificates for users or other certification authorities. They are organized hierarchically in what we call a PKI. Fig. 1 shows a PKI. The Root CA issues certificates to CA1 and CA2. These CAs issue certificates to CAs 1.1 and 1.2. Finally they issue certificates to the users (User 1 and 2).

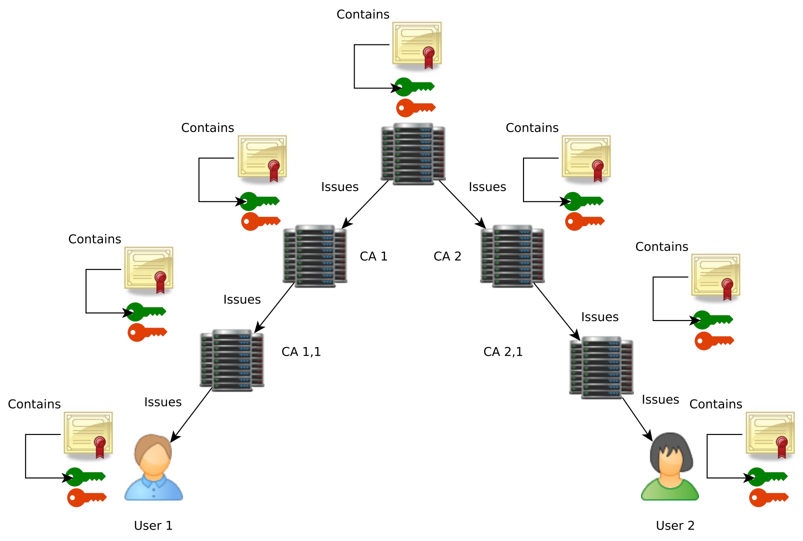

Figure 1: A PKI

3.5.4.1 Public Key Infrastructure (PKI) (cont'd)

There are two ways to enable the transferring of the trust: the Certificate chain and the Web of Trust.

- **Certificate Chain**: Imagine that User 1 and 2 need to talk. Nevertheless, they do not trust each-other. Now let us say that user 1 sends a signed message to user 2 and attaches his certificate. The problem is that user 2 does not trust the certificate of user 1 since it is issued by CA 1.1 (and user 2 has no idea about it). To solve this, user 1 must attach the whole certificate chain. More specifically, to transmit a message m to user 2, user 1 must send: $< m, \sigma_{User1}, cert_{user1}, cert_{CA1.1}, cert_{CA1} >$ ($\sigma_{pr_{user1}}$ is the signature that user 1 produced with his private key). Once user 2 receives the certificate chain she sees that the certificate of user 1 is certified by CA1.1. Since she does not trust CA1.1 she continues and sees that the certificate of CA1.1 is certified by CA1. The certificate of CA 1 is certified by the Root CA which she trust. As a result, she trusts the whole chain and thus, the certificate of the user and can now verify its signature. In this example, the Root CA is the **trust-anchor**.

Another method to establish trust is **PGP (Pretty Good Privacy)**. We will discuss this later in the course when we talk about Privacy. So for now no need to worry!

3.5.4.2 Public Key Infrastructure (PKI) (cont'd)

What happens when the private key of an entity get compromised (stolen)?. What we need to do is to invalidate the certificate of this entity. To do this we use Certificate Revocation Lists (CRLs). These lists contain the serial numbers of all the certificates that are not valid. Each CA published its own CRL. For example if the private key of User 1 was stolen, CA1.1 must issue a CRL that will contain it. So user 2 will have to examine this CRL before she decides to talk to User 1.

3.6 Session Key

During the recitations we mentioned that asymmetric cryptography is not efficient and thus, we use symmetric cryptography to encrypt large volumes of data. The entities must agree on these keys before they start communicating.

3.6.1 Key Agreement

The first key agreement we discussed was the Diffie–Hellman key exchange (D-H) key sharing algorithm. This algorithm allows two entities to establish a shared key by combining their secret pieces of information (without disclosing them!). During the recitations we went through the following example:

- Alice and Bob agree on $p = 23$ and $g = 5$.
- 2 Alice chooses $a = 6$ and sends $56\%23 = 8$.
- 3 Bob chooses $b = 15$ and sends $515\%23 = 19$.
- 4 Alice computes $196\%23 = 2$.
- 5 Bob computes $815\%23 = 2$.

The D-H is vulnerable to the Man-in-the-Middle attack since it does not authenticate the other end (we talked on how an attacker can establish 2 session keys with each of the entities and make them believe they communicate with each-other). Authentication or other variants are thus needed (we can use digital certificates and encrypt the values with public keys so that only the holders of the private keys can decrypt them).

3.6.2 Key Distribution center

Using a trusted service for getting the keys on demand. Alice ask a session key to the Key Distribution Center (KDC) encrypting the request $K_{A-KDC}(A, B)$ with the key $A - KDC$ shared with the KDC. The service replies with an encrypted response $K_{A-KDC}(R1, K_{B-KDC}(A, R1))$ containing the session key $R1$ in plaintext but also encrypted with the key $B - KDC$ shared between Bob and the KDC. Alice sends the encrypted message to Bob that is the only one able to decrypt it and read R1. We will revisit this protocols both during the next recitations and when we talk about unauthorized-access.

3.6.3 Transport Keys

If the two entities that want to communicate have certificates, we can use them to establish symmetric session keys. One entity can decide what symmetric key will be used and can encrypt it with the public key of the other. The other entity can decrypt by using its private key and gain access to the session key. There are different variants of this protocol that we will discuss in the next recitations.

[1] Vern Paxson. *Introduction to Communication Networks. Lecture 21*. EE122. Berkeley University. 2011. URL: http://inst.eecs.berkeley.edu/~ee122/fa11/.