Advanced Network Programming IV

Juliusz Chroboczek

24 March 2025

1 Introduction to security

The notion of *security* only makes sense within a certain context. For example, the web page with these lecture notes is not protected by a password; since these lecture notes are public, this does not constitute a security issue.

More precisely, the context of a security claim is defined by two sets of assumptions:

- the *security policy*, which described the desired properties (for example, the lecture notes are public but the exam subject is only available to the lecturer and the TAs);
- the *attack model*, which described the moves the attacker is capable of (for example, I assume that students might try to guess the password of a password-protected exam subject, but that they won't physically break into my home).

Of course, a large part of the job of securing a protocol consists in ensuring that the security policy and the attack model are realistic. Many attacks are due not to cryptographic or software flaws, but to an overly permissive security policy or an overly restrictive attack model.

2 Security properties

There are a number of standard security properties that we aim to satisfy.

Confidentiality Confidentiality is probably the most intuitive security property: it means that the attacker is unable to learn the content of the communication.

It used to be considered that confidentiality without authenticity is useless, as an attacker can mount a *Man in the middle* (MITM) attack, as described during the lecture. This point of view is no longer universal, as confidentiality without authentication ("ad-hoc encryption", or "better than nothing security") is fairly successful at thwarting mass surveillance.

Forward secrecy Forward secrecy is a particular case of confidentiality: it is confidentiality where the attacker is future me. Imagine an attacker (for example a security agency) that records encrypted traffic; the attacker might later obtain the encryption key, either by breaking into a home or by using the courts. Forward secrecy is that property that encrypted traffic cannot be decrypted in the future even if the attacker obtains the encryption key.

Authenticity Authenticity is the property that a message does come from the claimed sender. For example, the university will, someday, send a message to their bank asking to send me money; the bank will hopefully check the authenticity of the message in order to ensure that I (or anyone else) cannot fake the transfer order.

Integrity Integrity is the property that a message has not been modified between the sender and the receiver. Coming back to the previous example, the sum of money that the university orders to be transferred must remain unchanged between the sender (the university) and the receiver (the bank).

Availability Availability is the property that a service remains available even though the attacker is trying to make it fail. As we have seen during the lecture, there are a number of simple (but illegal[1]) methods that can make a network service unavailable; since there's not much that can be done, the security community sometimes neglects this property.

An attack against availability is called a *denial of service* (DoS). We have seen during the lecture specific cases of DoS, notably the *distributed denial of service* (DDoS) and the *reflection attack*.

3 Security primitives, constructions and protocols

Cryptographers provide us with *security primitives*, mathematical constructions that have some well-defined properties. Security primitives tend to be difficult to use, so cryptographers also provide us with *constructions*, built upon primitives, and easier to use. With these constructions, we build *security protocols*, which ensure that our network protocols satisfy the desired security properties (under a given attack model).

3.1 Primitives

A few examples of security primitives.

3.1.1 Block encryption

Given a key k, a block encryption is a bijective function E_k that maps a block of n bytes, the *cleartext*, to a block of n bytes, the *ciphertext*, where n is the *block size*. A block encryption function has the property that it is unfeasable to guess B given $E_k(B)$ without knowledge of k, even if the attacker has access to na large number of pairs $(B, E_k(B))$.

The current standard block encryption function is the *Advanced Encryption Standard* (AES), formerly known as *Rijndael*.

Block encryption is not directly usable by cryptographic protocols, for at least two reasons. First, block encryption operates on fixed-sized blocks, which is not flexible enough for applications. Second, block encryption maps two identical pieces cleartext to the same ciphertext, which gives too much information to the attacker.

1. Please don't get in trouble.

3.1.2 Cryptographic hash

A *hashing function* is function from sequences of bytes to a finite space:

$$h : A^* \longrightarrow 2^n$$

that is *statistically injective*: given $x \neq y$, it is probable (but not certain) that $h(x) \neq h(y)$. A pair x, y such that $h(x) = h(y)$ is called a *collision*.

A *cryptographic hashing function* is a hashing function for which we may assume that the attacker is unable to find a collision (it is reasonable to include this assumption in the attack model). The currently recommended cryptographic hash function is SHA-2, other choices include SHA-3 and the *Blake* family of functions.

Many cryptographic functions (including SHA-2) are vulnerable to *extension attacks*: given $h(A)$, it is possible to compute $h(A \cdot B)$, where $A \cdot B$ is the concatenation of A and B, without knowing A. This implies that cryptographic hash functions cannot be directly used for integrity protection; a construction is necessary.

3.2 Constructions

As we have seen above, most primitives are not suitable for use in cryptographic protocols. Cryptographers provide us with higher-level *constructions*.

3.2.1 HMAC

A Message Authentication Code (MAC) is a function

$$\mathrm{MAC}_k : A^* \longrightarrow 2^n,$$

where k is a secret, that can be used to verify the authenticity and integrity of messages: it is impossible to compute $\mathrm{MAC}_k(A)$ without knowning k, and if $\mathrm{MAC}_k(A) = \mathrm{MAC}_k(B)$, then it may be assumed that $A = B$.

Suppose that Alice and Bob have a shared secret k, and that Alice wants to send a message to Bob while ensuring authenticity and integrity. Naively, Alice might define

$$\mathrm{BADMAC}_k(m) = h(k \cdot m)$$

where h is a cryptographic hash function vulnerable to extension attacks. Alice sends the pair

$$(m, \mathrm{BADMAC}_k(m))$$

to Bob. Bob verifies the message by recomputing the MAC and comparing it to the received MAC.

Damian, the system administrator, wants to annoy Bob[2]. Since Damian has access to the network routers, he intercepts the message and appends a post-scriptum:

$$m' = m \cdot p$$

Since h is vulnerable to extension attacks, Damian can compute $h(k \cdot m \cdot p) = \mathrm{BADMAC}_k(m')$, Bob will verify the MAC, and assume the message is authentic and integrity-protected.

An HMAC is MAC construction based on a cryptographic hash function that is invulnerable to such extension attacks. HMAC is defined in RFC 2104.

2. Bob uses a Mac.

3.2.2 Symmetric encryption with a mode of operation

As we have seen above, block encryption is not suitable for direct usage by cryptographic protocols, for at least two reasons. In order to provide useful constructions, cryptographers combine block encryption with *modes of operation* in order to provide encryption algorithms that work on streams of arbitrary length and hide repeated data within a stream (but not necessarily between streams.

There is a whole fauna of modes. As far as I am aware, only two modes are currently used in network protocols:

- *Counter mode* (CTR), which is simple to ipmlement and can be parallelised;
- *Galois/Counter mode* (GCM), which is more complex but provides both encryption and integrity protection.

4 Protocols

Cryptographic constructions can be directly used in network protocols; in practice, however, it is often the case that a standard cryptographic protocol is available that solves part of the security needs of the application. When that is the case, it is essential for the application designer to have a full understanding of the attack model and the security properties assumed by the cryptographic protocol (and its implementation).

In the first part of this course, we will use a standard security protocol, TLS. In the second part, we will implement our own security protocols based on standard constructions.

4.1 TLS and HTTPS

TLS is a standard protocol, used notably in HTTP. The combination of TLS and HTTP is called HTTPS. TLS has the following properties:

- it can only protect streams of bytes, for example TCP streams (a variant called DTLS can protect client-server streams of datagrams);
- it is restricted to client-server communication;
- it provides confidentiality;
- forward secrecy is optional in TLS 1.2, and mandatory in TLS 1.3;
- authentication is optional.

When used in HTTPS, TLS has the following additional restriction:

- HTTPS authenticates the server, but not the client.

We will manipulate TLS in the next lab session, and explore the notion of *X.509 certificate*, which is a data structure that is used to authentify the HTTP server.

4.2 Username/password authentication

The profile of TLS used in HTTPS authenticates the server only; thus, there is a need for a mechanism to authenticate the client.

The most common such mechanism is *username/password authentication*. The server has access to a database of user accounts, where each row contains (among other data) a username and a password. When a client connects, it sends a username/password pair to the server, which verifies the password against the database.

Username/password authentication has a number of weaknesses. The most significant is that users tend to use the same password on multiple servers, which implies that a password leak has serious consequences. As the server is typically maintained by underpaid interns who have had one semester of PHP programming, password leaks are a serious issue.

Failure of other mechanisms There are a number of other mechanisms that have been proposed to replace username/password authentication, but they have all failed to gain acceptance. With all their flaws, passwords are a familiar concept, that users have accepted, and in practice it is difficult to impose a more secure mechanism.

4.3 Token authentication

Token authentication is an improvement over username/password which aims to minimise the likelihood of password leaks. In token authentication, there are two servers, the *authentication server* and the *application server*. The authentication server has access to the password database, and both the authentication server and the application server share a token database, initially empty.

When a user connects to the service, the client first sends a request to the authentication server, which consults the password database, and, if the user is authorised, generates a *token*, an opaque string of bytes. It inserts a row in the token database, that contains the token, and any other data required to use the service (such as the username, the user id, the user's permissions, etc.).

The client then connects to the application server, and authentifies using the token (the password is never sent to the application server). The application server consults the token database, and if the token is found, it grants access to the client.

The main advantage of this mechanism that only the authentication server (written in Rust by a security expert) has access to the password database; the application server (written in PHP or Python by an underpaid intern) only has access to the token database; in the case of a vulnerability, only the token database is leaked, which is easily revocable. Token authentication is used by a number of currently fashionable protocols, notably *OAuth2* and *OpenID Connect*.

There exists a variant of this technique, using *cryptographic* or *stateless* tokens, which avoids the need for a token database. We shall explore stateless tokens during the lab.