MTLS
Doble autenticación con Springboot y Nginx

Paradigma

Contenido

1. Cifrado simétrico y asimétrico
2. Certificados X.509 (Que son, infraestructura, formatos)
3. SSL/TLS
4. MTLS
5. Implementación de MTLS

Cifrado simétrico y asimétrico

El **cifrado simétrico**: Solamente utiliza una clave, es el más simple y rápido de los dos, desventaja es menos seguro.

El **cifrado asimétrico**: Solamente **utiliza dos claves**, una de ellas es privada y otra es pública, es menos simple y rápido pero es más seguro.

¿Que es un certificado X.509?

“Es un estándar para infraestructuras de clave pública, y específica formatos estándar para certificados de clave pública.”

*“Un certificado X.509 es una estructura de datos en el que se produce una asociación entre los **datos de un titular** que aseguran su identidad y una **clave pública**.*

La asociación es realizada por una entidad confiable mediante una firma digital, con lo que es posteriormente verificable.”

Paradigma

Infraestructura de clave pública

Autoridad de Certificación (CA)

- Emite certificados.
- Emite listas de certificados revocados.

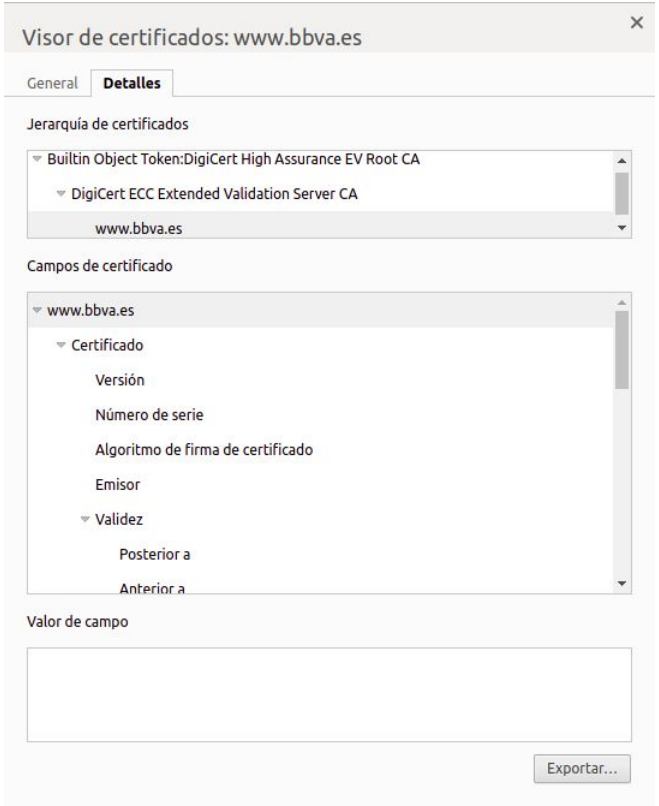
Autoridad de Registro (RA)

- Recoge las solicitudes/peticiones de certificado y comprueba al peticionario para enviarlas luego a la CA.

Autoridad de Validación (VA)

- Responde a las peticiones de consulta del estado de revocación de los certificados. Protocolo OCSP.

Infraestructura de clave pública



Autoridad de certificación

La autoridad de certificación firma un certificado con el cual firma las peticiones de certificado que llegan.

TIpos:

- **Autoridad de certificación root**, punto principal de confianza, tiene un certificado autofirmado.
- **Autoridad de certificación subordinada**, Es emitida por la root o por otra autoridad subordinada.

Paradigma

Formatos

PKCS#10 Petición de certificado (CSR).

DER Codificación en binario de la estructura ASN.

PEM El DER codificado en B64 y delimitado por cabeceras:

```
----------- BEGIN CERTIFICATE ------------
<Certificado en B64>
------------- END CERTIFICATE ------------
```

PKCS#12 Certificado más la clave privada protegido con un password.

PKCS#7 Puede ser cualquier cosa, por ejemplo una lista de certificados.

TLS/SSL

Son protocolos criptográficos, que proporcionan comunicaciones seguras en internet.

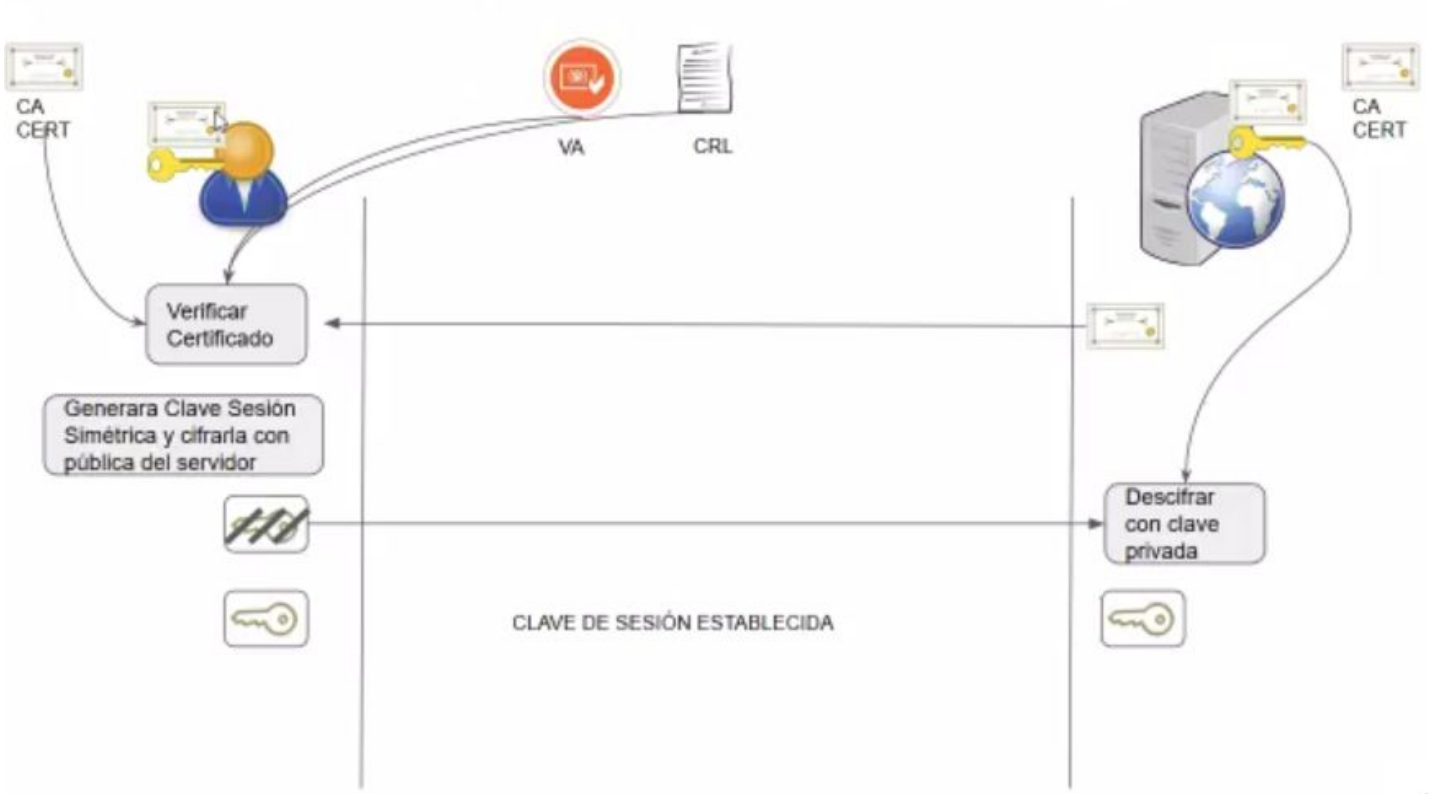
Este protocolo usan **certificados X.509** y por lo tanto criptografía asimétrica (es decir, con clave pública y privada).

Permite y garantiza el intercambio de datos en un entorno securizado y privado entre el cliente y el servidor.

TLS: Seguridad de la capa de transporte, en inglés Transport Layer Security. (1999)

SSL: Capa de puertos seguros, en inglés Secure Sockets Layer, **su antecesor.** (1995)

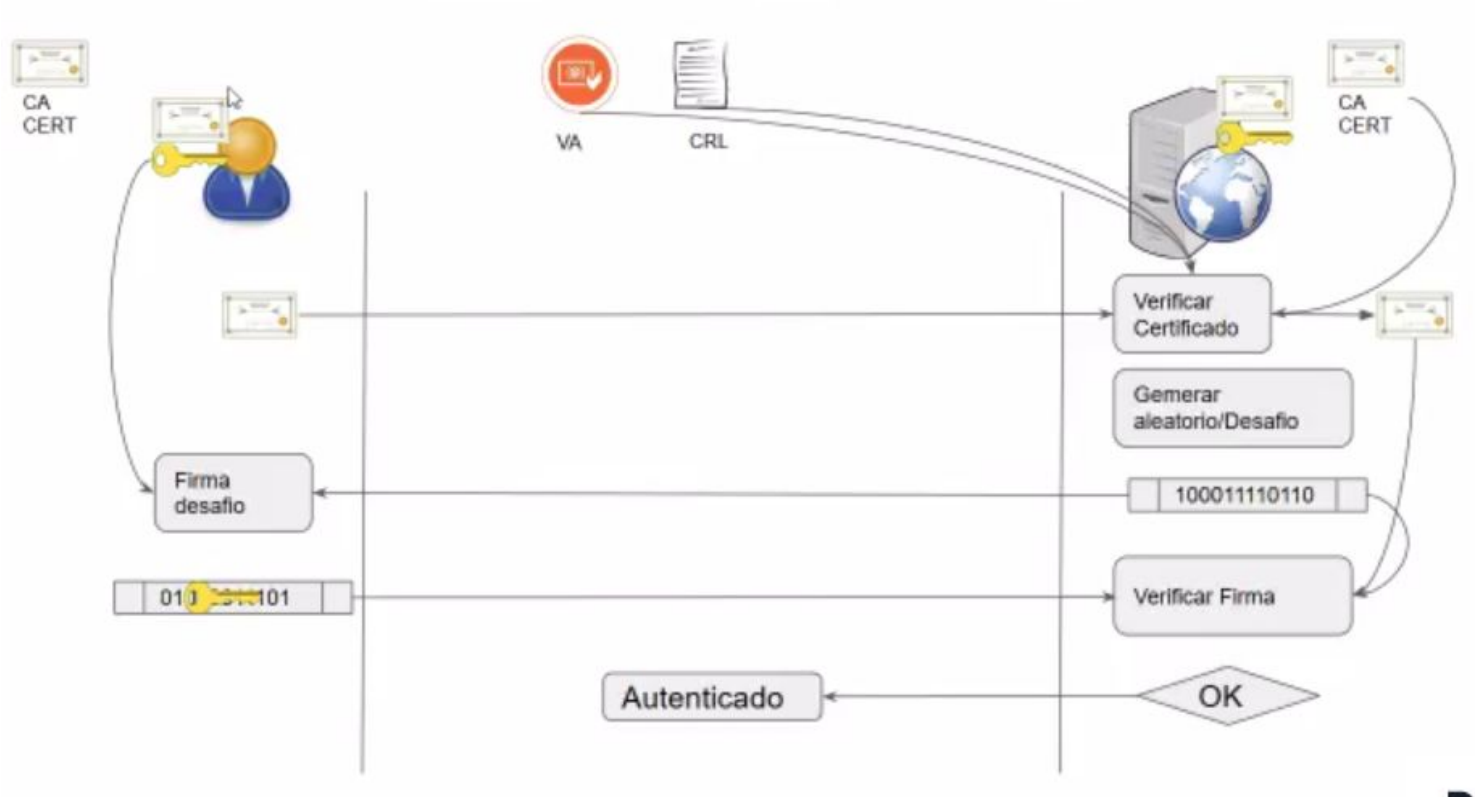
SSL/TLS (Certificado x.509 - Cifrado Asimétrico)



Fuente:BBVA

Paradigma

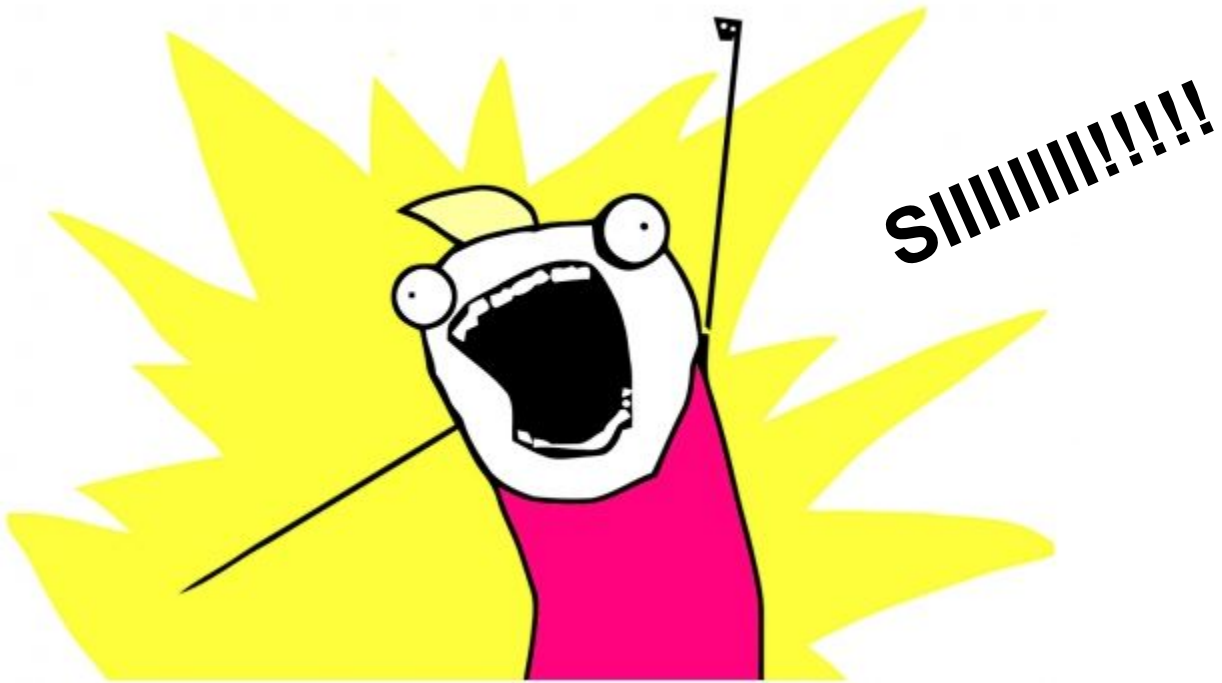
Mutual TLS (Certificado x.509 - Cifrado Asimétrico)



Fuente:BBVA

Paradigma

Queremos código...



Implementación en Servidor (Nginx)

Definición del Keystore y del Truststore (nginx.conf):

```
listen 443 ssl default_server;
ssl_certificate        /etc/nginx/ssl/myserver.crt;
ssl_certificate_key    /etc/nginx/ssl/myserver.key;
#ssl_protocols       TLSv1 TLSv1.1 TLSv1.2; #TLSv1.3

ssl_client_certificate  /etc/nginx/ssl/ca_demo_client.crt;
ssl_verify_depth 10;  # 1
ssl_verify_client optional;  # on | off | optional |
optional_no_ca
...
proxy_set_header vnd.demo.certificate.subject $ssl_client_s_dn;
proxy_set_header vnd.demo.certificate.issuer $ssl_client_i_dn;
proxy_set_header vnd.demo.certificate.verify $ssl_client_verify;
# “SUCCESS”, “FAILED:reason”, and “NONE”
```

ca_demo_client.crt (PEM format)

```
-----BEGIN CERTIFICATE-----
MIIG5zCCBM+gAwIBAgITJgcgExc02DzTuG79IQAAAAAACDANBgkqhkiG9w0BAQsF
ADBnMQswCQYDVQQGEwJFUzENMAsGA1UECgwEQkJWQTErMCkGA1UECwwiU2VjdXJp
...
dHkgQXJjaGl0ZWN0dXJlIENyeXB0b2dyYXBoeTEcMBoGA1UEAwwTR2xvYm==
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIG5zCCBM+gAwIBAgITJgcgExc02DzTuG79IQAAAAAACDANBgkqhkiG9w0BAQsF
BgNVBAYTAkVTMQ0wCwYDVQQKDARCQlZBMSswKQYDVQQLDCJTZWN1cml0eSBBcmNo
...
b3QgQ0EgV29yazAeFw0xODA2MjUxMjAwMDBaFw0yODA2MjUxMjAwMDBaMHkxCz=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIG5zCCBM+gAwIBAgITJgcgExc02DzTuG79IQAAAAAACDANBgkqhkiG9w0BAQsF
aXRlY3R1cmUgQ3J5cHRvZ3JhcGh5MS4wLAYDVQQDDCVHbG9iYWwgSXNzdWluZy
...
BgNVBAYTAkVTMQ0wCwYDVQQKDARCQlZBMSswKQYDVQQLDCJTZ=
-----END CERTIFICATE-----
```

Paradigma

Implementación en Servidor (Springboot)

La verificación del backend la podemos hacer de varias maneras, la más simple, implementamos una clase FIlter, extraemos las cabeceras que nos manda el Nginx y las validamos contra una BBDD, propiedades de configuración, etc.

```
String issuer = request.getHeader(HEADER_CERTIFICATE_ISSUER);

String subject = request.getHeader(HEADER_CERTIFICATE_SUBJECT);

String verification = request.getHeader(HEADER_CERTIFICATE_VERIFY);
```

Implementación de cliente (Springboot-Apache HttpClient)

Paso 1 - Creamos un SSLContextBuilder customizado

```
SSLContextBuilder SSLBuilder = SSLContexts.custom();
```

Paso 2 - Cargamos el Truststore (Cargo todas las CAs en las que confio)

```
File file = new File("mytruststore.jks");
SSLContext sslContext = SSLBuilder.loadTrustMaterial(file, "changeit".toCharArray()).build();
```

Paso 3 Opción 1 - Cargamos el Keystore (Cargo los certificados de un keystore ya creado)

```
sslContextBuilder =  SSLBuilder.loadKeyMaterial(ResourceUtils.getFile("classpath:keystore.jks"),
password.toCharArray(), alias.toCharArray());
```

Paso 3 Opción 2 - Cargamos el Keystore (Creo un keystore “on-the-flight” con un PKCS12 y le añado un certificado)

```
KeyStore keyStore = KeyStore.getInstance("PKCS12");
FileInputStream keyStoreInput = new FileInputStream(keyStorePath);
keyStore.load(keyStoreInput, keyStorePassword.toCharArray());
sslContextBuilder = SSLBuilder.loadKeyMaterial(keyStore, keyStorePassword.toCharArray());
```

Implementación de cliente (Springboot-Apache HttpClient)

Paso 4 - Creamos un HttpClient

```
SSLContext sslContext = SSLBuilder.build();
HttpClient client = HttpClients.custom().setSSLContext(sslContext).build();
```

Paso 4 - Creamos un RequestFactory

```
HttpComponentsClientHttpRequestFactory requestFactory =
      new HttpComponentsClientHttpRequestFactory();
requestFactory.setHttpClient(client);
```

Paso 5 - Creamos un RestTemplate

```
RestTemplate restTemplate = new RestTemplate(requestFactory);
```

Paso 6 - Ejecutamos la request

```
ResponseEntity<String> result = restTemplate.exchange(url, httpMethod, requestEntity,
String.class);
```

Ejemplo práctico 1

Usando un servidor (Nginx, Springboot y Docker)

Usando un cliente Java (Springboot + Spring Web + Apache HttpClient)

...Tenemos bola extra!!!

Implementación en Servidor (Spring Security & Tomcat)

En el configuration:

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
       http.authorizeRequests()
          .anyRequest().authenticated().and().x509()
          .subjectPrincipalRegex("CN=(.*?)(?:,|$)")
          .userDetailsService(userDetailsService());
       }
    @Bean
    public UserDetailsService userDetailsService() {
       return (UserDetailsService) username -> {
           if (username.equals("hugo")) {
              return new User(username, "",  AuthorityUtils
               .commaSeparatedStringToAuthorityList("ROLE_USER"));
           } else {
              throw new UsernameNotFoundException();
           }
       };
    }
}
```

En nuestro Controller:

```
@GetMapping("/{id}")
@Secured("ROLE_USER")
public Customer GetCustomer(@PathVariable Long id) {
   return new Customer(id, "Customer" + id);
}
```

En el application:

```
server.port=8443
server.ssl.key-store-type=PKCS12
server.ssl.key-store=classpath:myserver.p12
server.ssl.key-store-password=pwd0
server.ssl.trust-store=classpath:mytruststore.jks
server.ssl.trust-store-password=changeit
server.ssl.trust-store-type=JKS
server.ssl.client-auth=need
```

Paradigma

Ejemplo práctico 2

Usando un servidor (Springboot y Spring Security)

Usando un cliente cURL

```
curl -k -X GET \
  --key ~/Documentos/MIGUEL/personal/cursos/docker/docker-images/alpine-nginx/ssl/client1.key \
  --cert ~/Documentos/MIGUEL/personal/cursos/docker/docker-images/alpine-nginx/ssl/client1.crt \
  https://localhost:8443/server/verifications/3456-v
```

Enlaces de interés

http://nginx.org/en/docs/http/configuring_https_servers.html

https://dzone.com/articles/securing-rest-apis-with-client-certificates

https://www.tutorialspoint.com/apache_httpclient/apache_httpclient_custom_ssl_context.htm

http://www.criptored.upm.es/intypedia/docs/es/video9/DiapositivasIntypedia009.pdf

https://gist.github.com/dain/29ce5c135796c007f9ec88e82ab21822


Thank You
That will be all for today