

UT5-TÉCNICAS DE PROGRAMACIÓN SEGURA

Programación de Servicios y Procesos

OBJETIVOS

- Comprender y aplicar los conceptos de
 - Criptografía simétrica o de clave privada
 - Criptografía asimétrica o de clave pública
 - Funciones resumen (digest) y hashes
 - Firma digital
 - Certificados digitales
- Aplicarlos al control del acceso de una API REST

CRIPTOGRAFÍA Y CIDAN

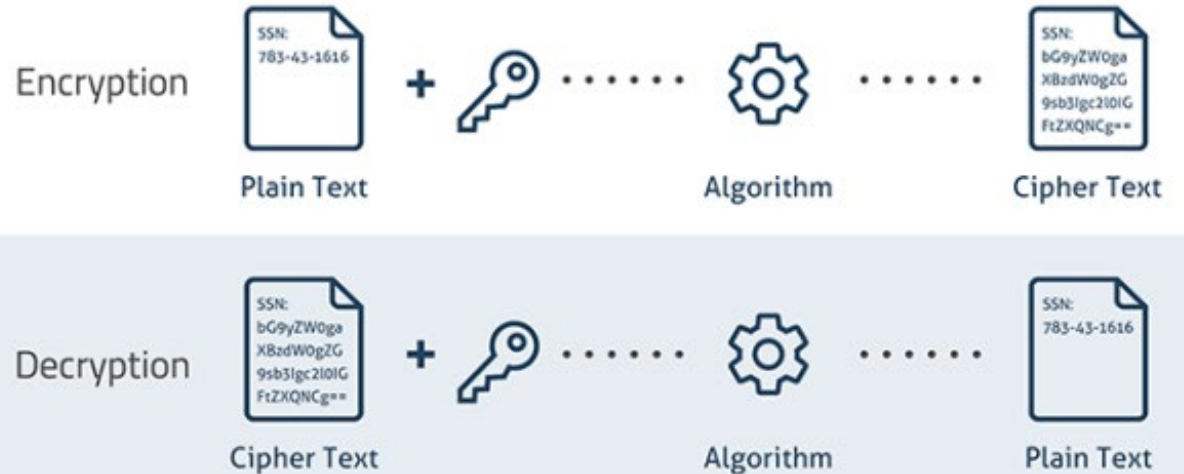
- **Confidencialidad**
 - Solo quien tiene permiso puede leerlo ✓
- **Integridad**
 - Si se modifica el mensaje se vuelve indescifrable ✓
- **Disponibilidad** ✗
- **Autenticación**
 - Se sabe quién es el originante ✓
- **No repudio** (en origen)
 - El originante no puede negar que ha cifrado ✓
- En resumen: podemos transmitir información de forma segura a través de un medio inseguro



CIFRADO: ELEMENTOS INVOLUCRADOS

- El mensaje
 - Un conjunto de bits (que pueden ser texto por ejemplo).
- La clave
 - Un conjunto de bits.
- El algoritmo
 - Una serie de pasos en programación, como por ejemplo esta [función](#).

SAMPLE ENCRYPTION AND DECRYPTION PROCESS

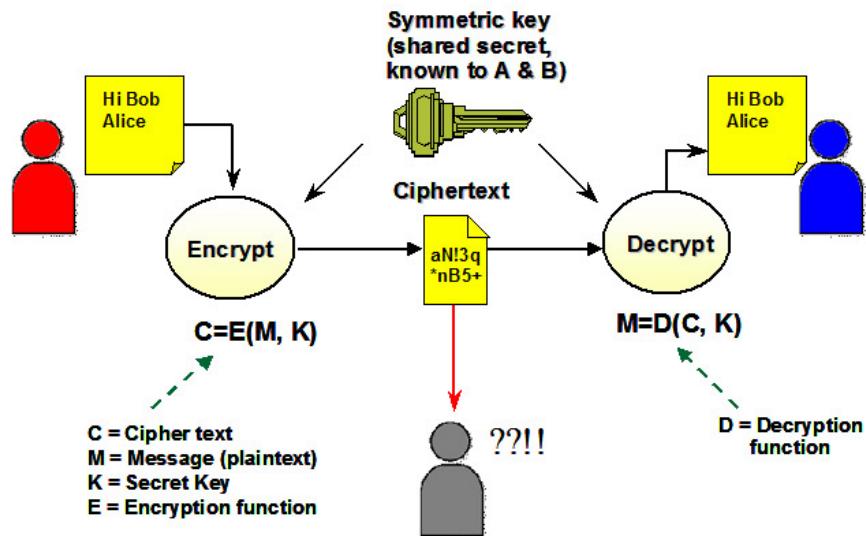


CRIPTOGRAFÍA EN JAVA

- JCA: Java Cryptography Architecture. Paquete `java.security`
 - Funciones resumen (digest)
 - Generación de claves y pares de claves
 - Almacén de claves
 - Certificados
- JCE: Java Cryptography Extensions. Paquete `javax.crypto`
 - Cifrado
 - Generación de MAC

CRIPTOGRAFÍA SIMÉTRICA

CRIPTOGRAFÍA DE ALGORITMOS SIMÉTRICOS O DE CLAVE PRIVADA



Simétricos

Utilizan la misma clave para cifrar y descifrar el mensaje.

Clave privada

La clave solo deben conocerla el emisor y el receptor.

CRIPTOGRAFÍA DE ALGORITMOS SIMÉTRICOS O DE CLAVE PRIVADA

DES (clave de 56 bits).

- Hay 72.057.594.037.927.936 combinaciones posibles.
- Un ordenador puede atacar por fuerza bruta en poco tiempo

3DES (clave 168 bits)

- [CVE-2016-2183](#) → Será deprecada por el NIST en 2023

Blowfish (32 a 448bits)

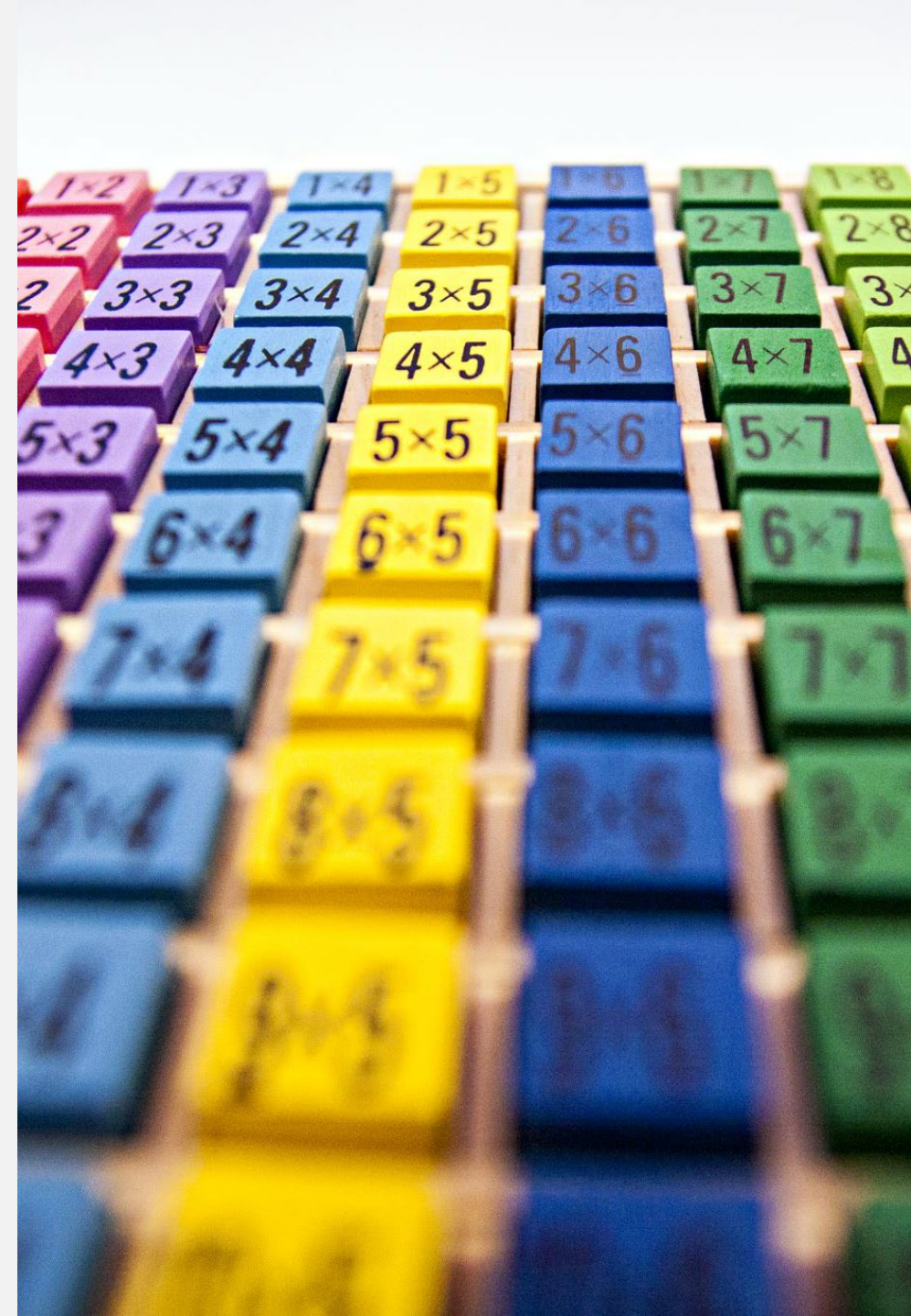
IDEA (128 bits)

AES (128, 192 y 256 bits) (Rijndael)

- El algoritmo más utilizado en la actualidad

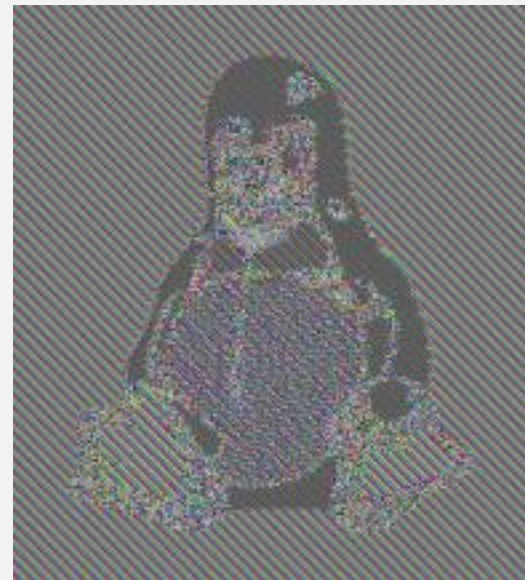
AES

- Claves de 128, 192 y 256 bits (16, 24 y 32 bytes)
- Cifra y descifra en bloques de 128 bits (16bytes)
- Diversos modos distintos de funcionamiento
- ¿Y cuál elijo? Pues [depende](#)

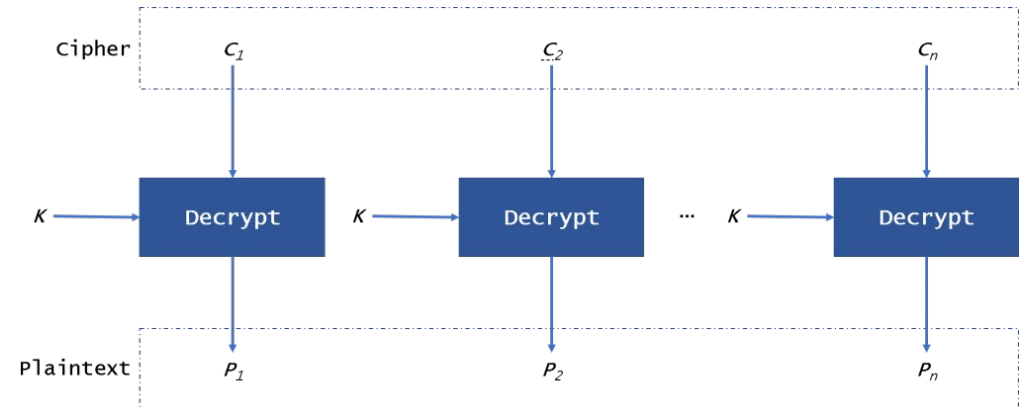
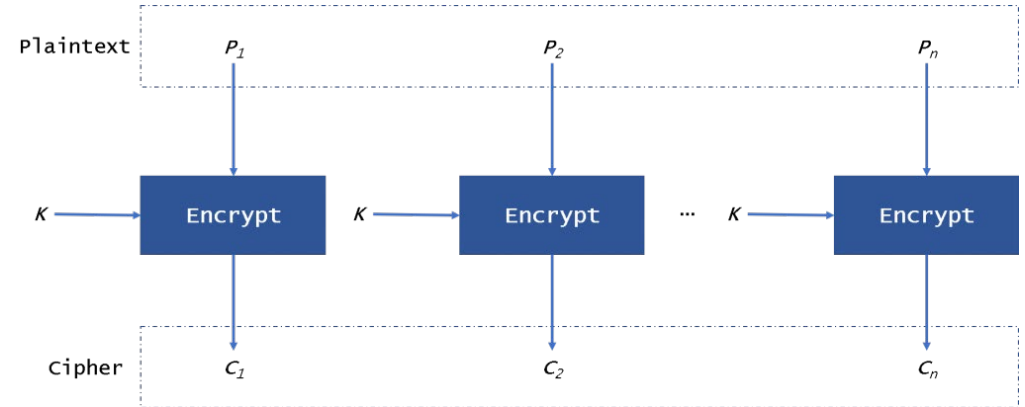


ECB

- Electronic Code Book
- Cifra cada bloque de la misma forma
- NUNCA utilizarlo para cifrar más de un bloque
- Requiere relleno



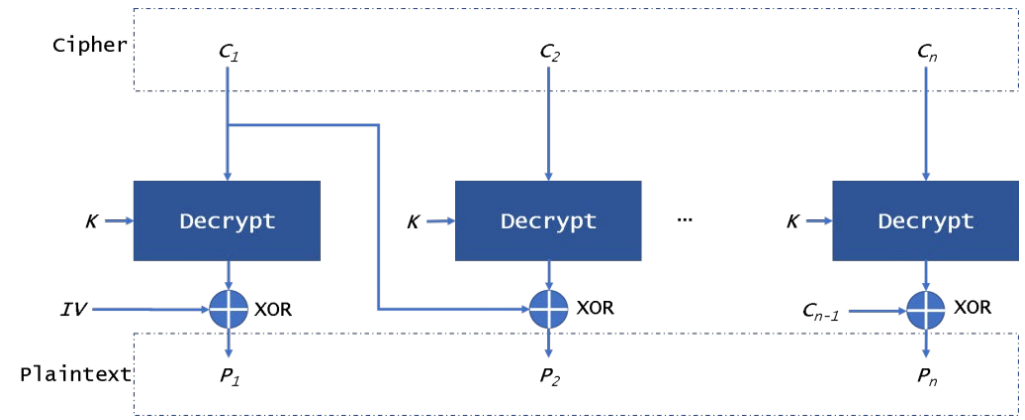
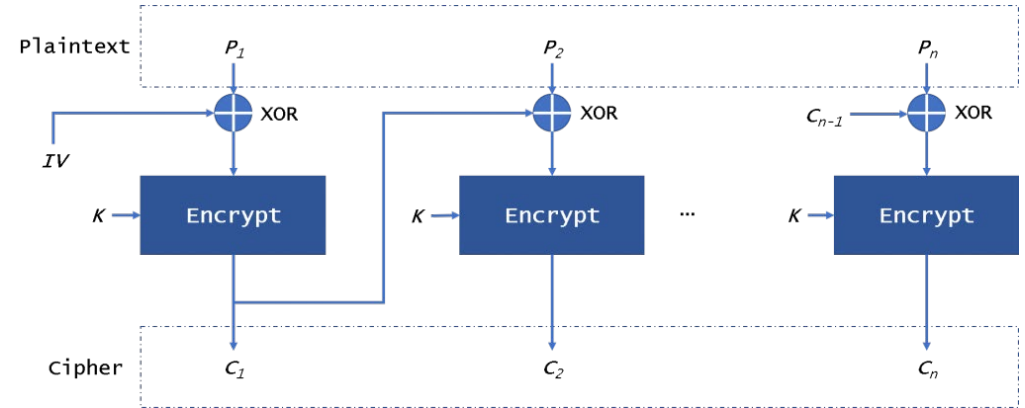
ECB



CBC

- Cipher Block Chaining
- Primer bloque cifrado con XOR de Vector de Inicialización (IV) y texto en claro.
- Bloques consecutivos, XOR del bloque cifrado previo y el texto plano.
- Requiere relleno.
- Permite descifrado en paralelo.

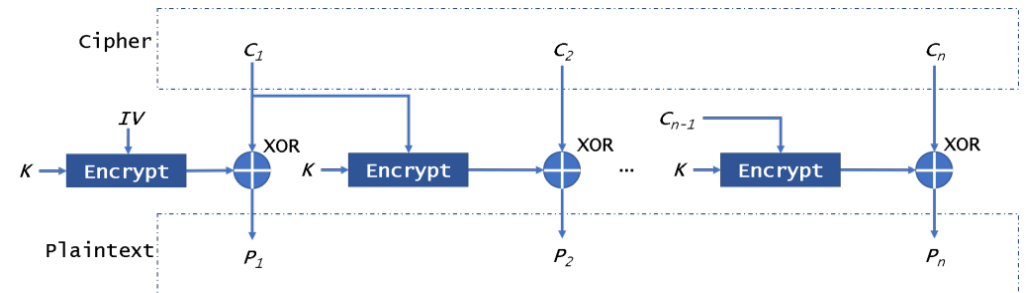
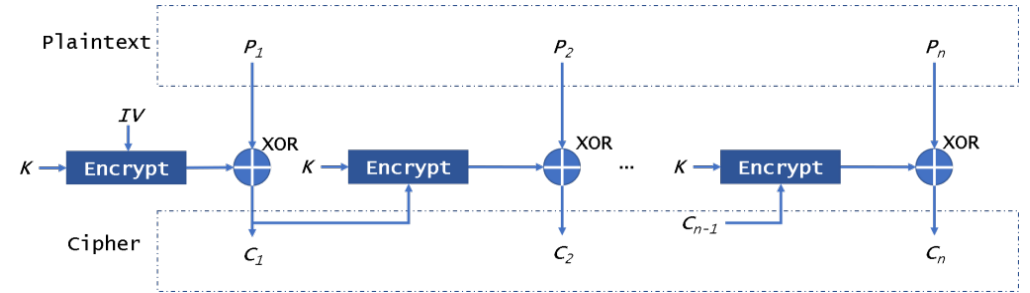
CBC



CFB

- Cipher FeedBack
- El primer bloque se obtiene cifrando el IV y haciendo XOR con el texto plano.
- Siguientes bloques cifra el resultado anterior y hace XOR con texto plano
- Permite descifrado en paralelo.
- Permite cifrar un flujo

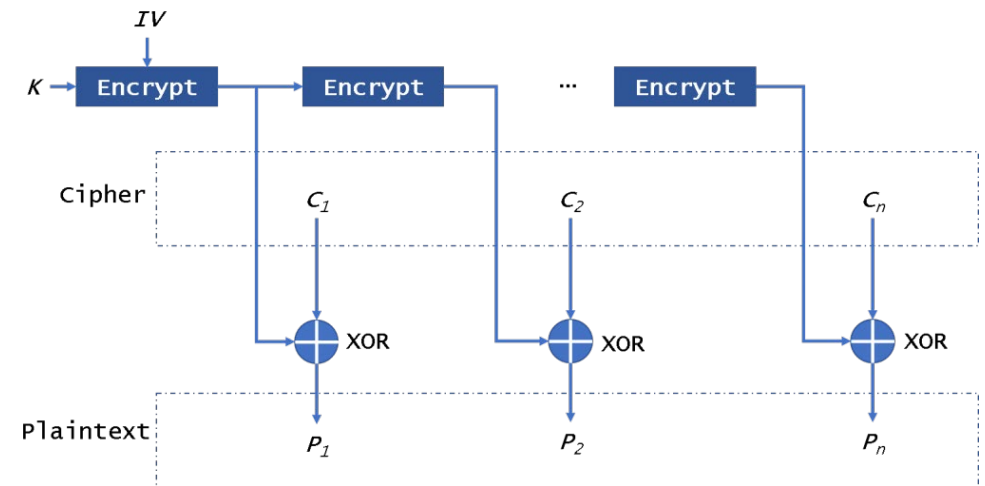
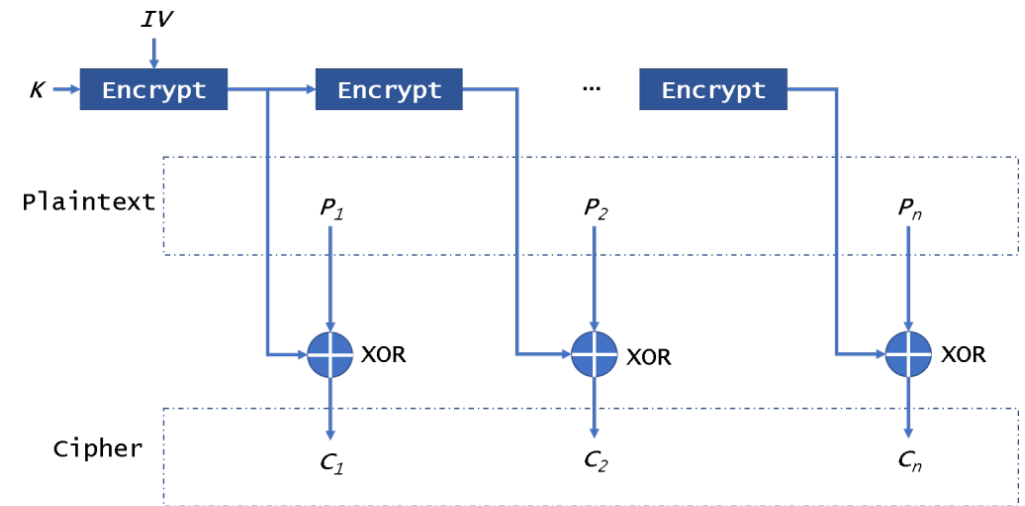
CFB



OFB

- Output FeedBack
- En cada bloque se vuelve a cifrar el IV y se realiza la XOR con el texto plano
- No permite ni cifrado ni descifrado en paralelo.
- Permite cifrar un flujo

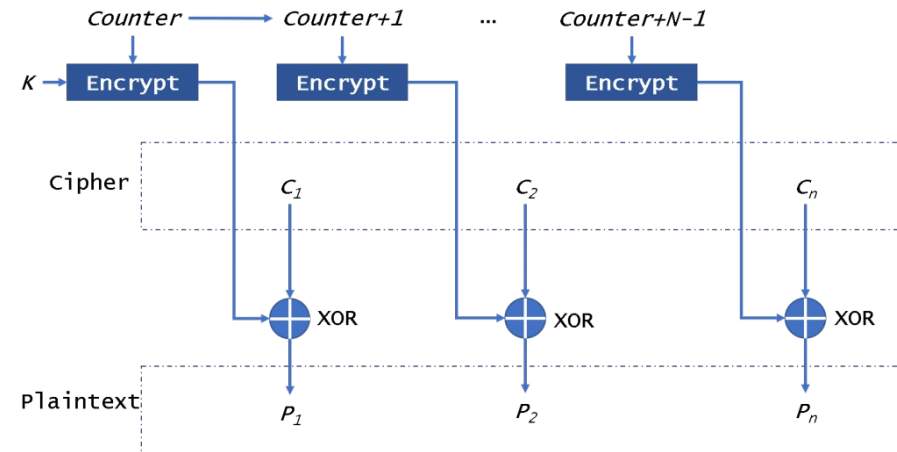
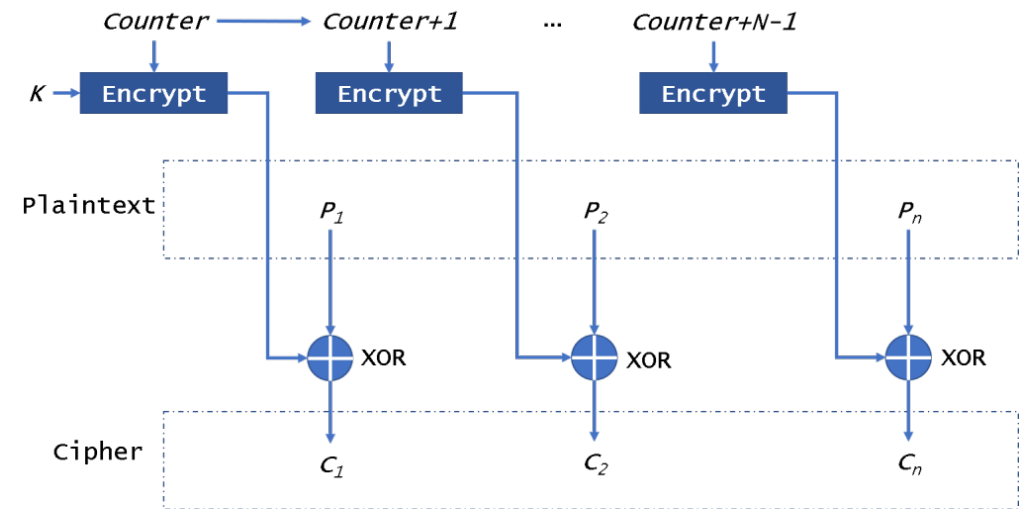
OFB



CTR

- Counter
- En cada bloque cifra el resultado de incrementar un contador y se hace la XOR con el texto en claro.
- Permite cifrado y descifrado en paralelo.
- Permite cifrar un flujo.

CTR



COMPARACIÓN MODOS

STREAM

- El texto cifrado tiene el mismo tamaño que el original
- Apropiado cuando el padding molesta
- CFB
- OFB
- CTR

BLOQUE

- El tamaño del texto cifrado es siempre un múltiplo de 128bits
- ECB
- CBC

¿PERO CÓMO LO HAGO CON JAVA?

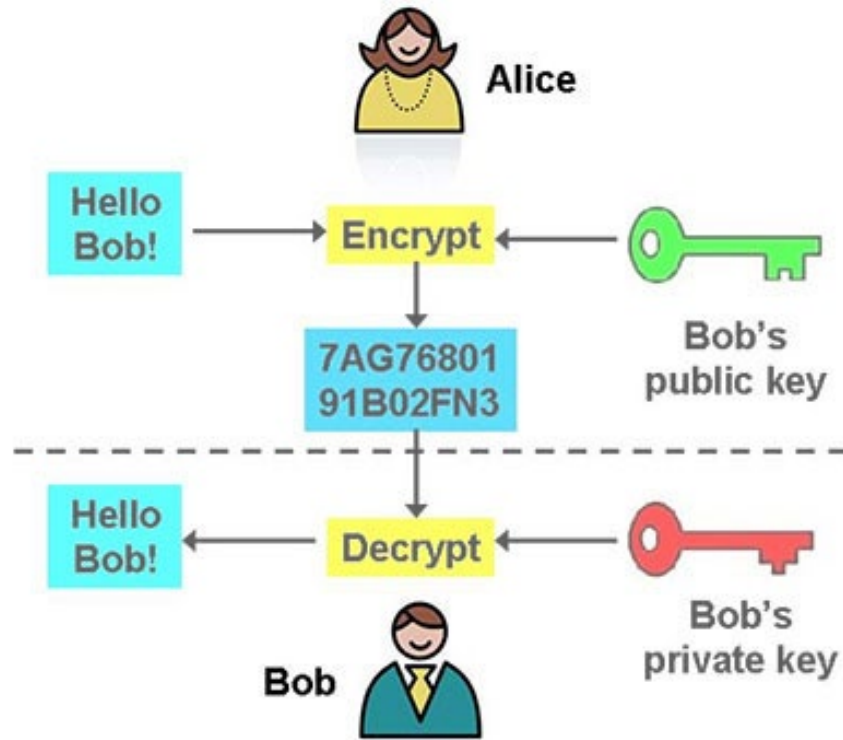
1. Generar una clave de cifrado `SecretKey`
 1. de forma aleatoria `KeyGenerator`
 2. o a partir de una contraseña `SecretKeyFactory`
2. Crear un vector de inicialización `IvParameterSpec` (salvo para ECB)
3. Cifrar una cadena o un fichero `Cipher`
 - Para los identificadores a utilizar ver [Java Security Standard Algorithm Names](#)
 - Ver `AesEncryption.java`

¿QUÉ TENGO QUE RECORDAR?

- Siempre debo utilizar clases del JDK, nunca implementes algoritmos criptográficos por tu cuenta
- Que necesito generar clave y vector de inicialización
- Utilizo la clase AesEncryption del ejemplo como plantilla
- Que el cifrado no solo requiere de un algoritmo como AES, sino una transformación que incluye algoritmo, modo de funcionamiento y padding
- Que cada modo de operación tiene ventajas e inconvenientes descritos en la web del [NIST](#). Presta atención a las recomendaciones de uso de los vectores de inicialización
- Que los nombres utilizados por los métodos factoría están en la web de [Oracle](#) para cada versión de Java

CRIPTOGRAFÍA ASIMÉTRICA

CRIPTOGRAFÍA DE ALGORITMOS ASIMÉTRICOS O DE CLAVE PÚBLICA



Asimétricos

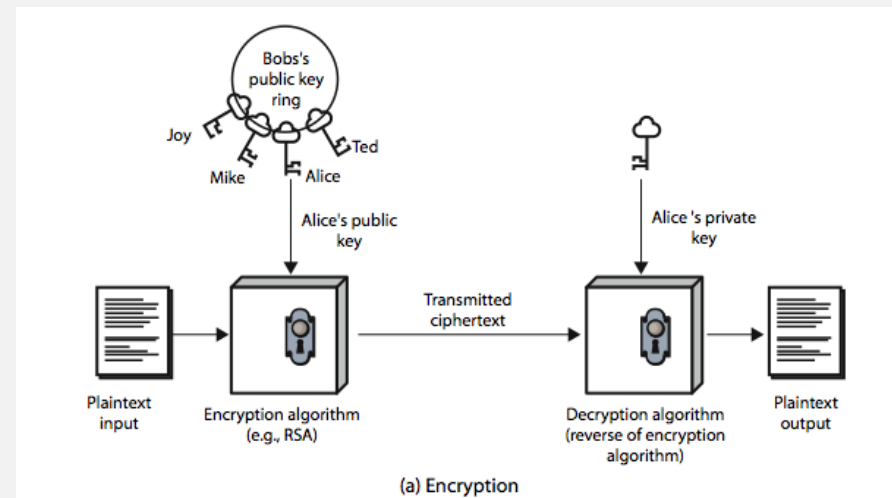
Cada interlocutor tiene un par de claves: lo que cifra una SOLO se puede descifrar con la otra.

De clave pública

Dentro de cada par llamamos a una “clave privada” y a la otra “clave pública”.

CRIPTOGRAFÍA DE ALGORITMOS ASIMÉTRICOS O DE CLAVE PÚBLICA

- ¿Y qué hago con las claves?
- La clave privada me la guardo.
- La clave pública la distribuyo a todo el mundo.
- ¿Qué pasa si alguien usa mi clave pública?
- ¿Qué pasa si alguien me roba mi clave privada?





ALGORITMOS DE CLAVE PÚBLICA

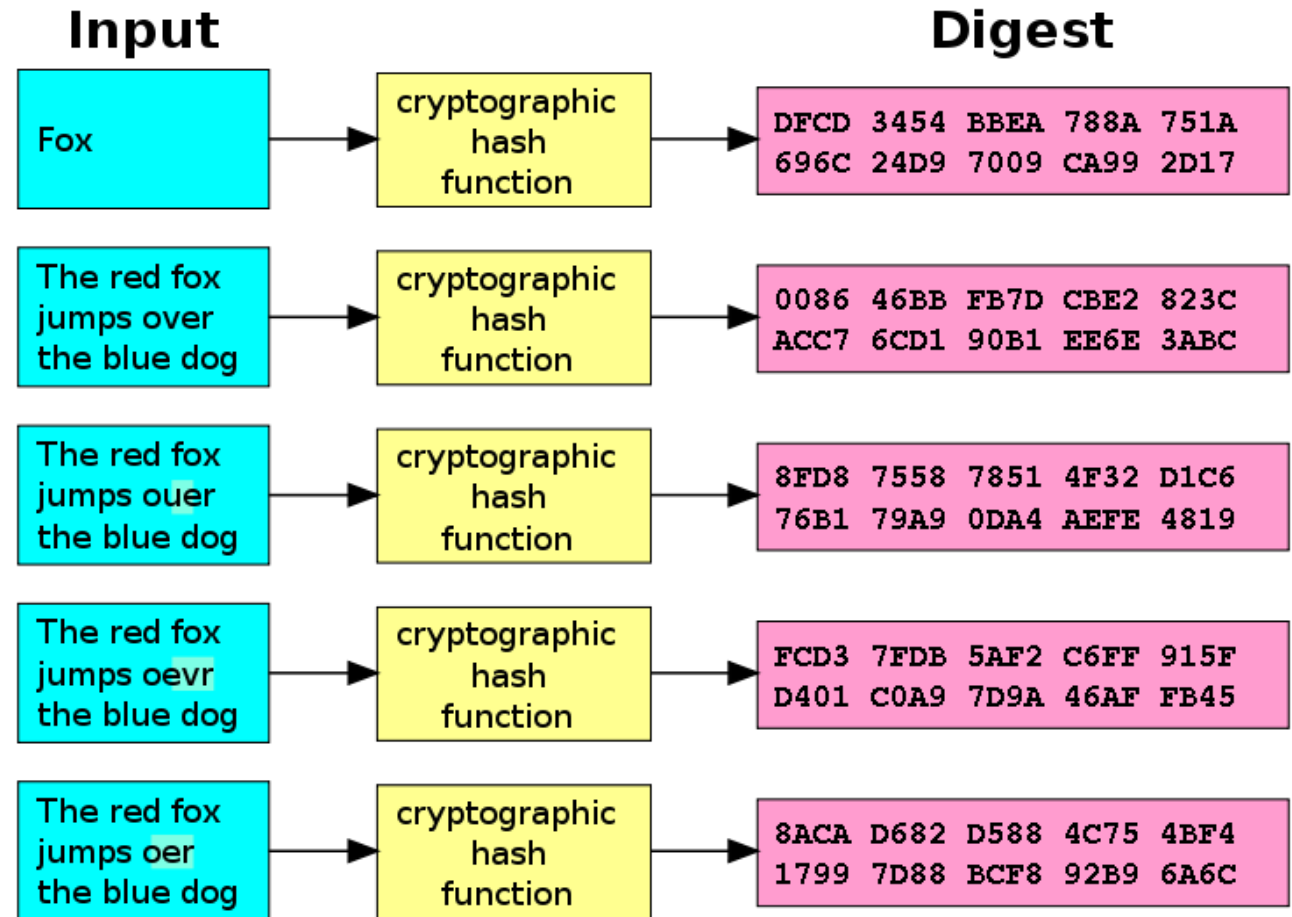
- DSA (Digital Signature Algorithm), se utiliza para firmar no para cifrar. Utiliza claves de 1024 bits.
- RSA (Rivest Shamir Adleman), un algoritmo antiguo basado en la dificultad de factorizar números grandes. Se recomienda un tamaño de clave de al menos 2048 bits para RSA; 4096 bits es mejor. RSA está envejeciendo y se están logrando avances significativos en factorización. Es muy posible que el algoritmo RSA se vuelva prácticamente rompible en un futuro previsible.
- ECDSA (Elliptic Curve Digital Signature Algorithm): un nuevo algoritmo de firma digital estandarizado por el gobierno de los EE. UU. que utiliza curvas elípticas. Este es probablemente un buen algoritmo para las aplicaciones actuales. Solo se admiten tres tamaños de clave: 256, 384 y 521 bits. Recomendamos usarlo siempre con 521 bits, ya que las claves aún son pequeñas y probablemente más seguras que las claves más pequeñas (aunque también deberían ser seguras que otros algoritmos más antiguos).

¿PERO CÓMO LO HAGO CON JAVA?

1. Generar un par de claves de cifrado `KeyPair` de forma aleatoria con `KeyPairGenerator`
2. Cifrar una cadena o un fichero `Cipher` con `PublicKey` o con `PrivateKey`
 - Para los identificadores a utilizar ver [Java Security Standard Algorithm Names](#)
 - Ver `RsaEncryption.java`

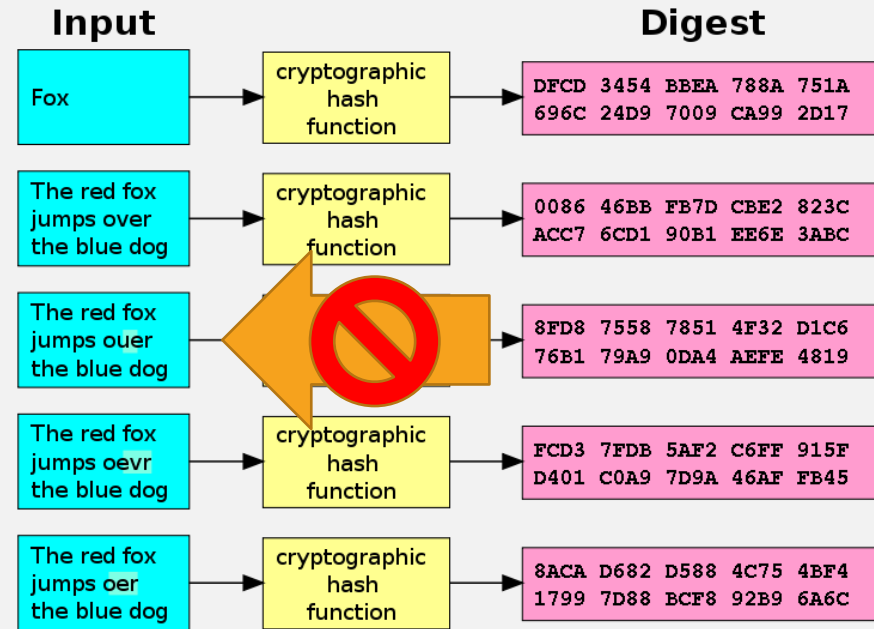
FUNCIONES HASH

FUNCIONES HASH O RESUMEN



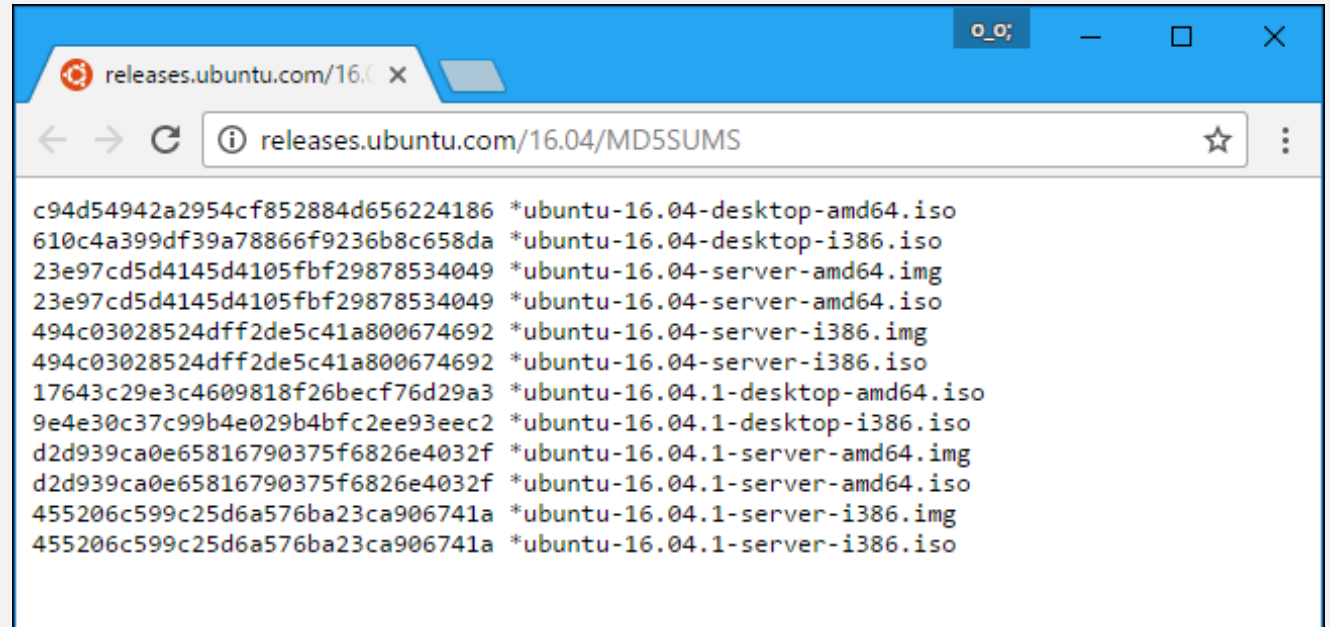
CARACTERÍSTICAS

- Ante la misma entrada producen siempre la misma salida.
- A partir de la salida NO se puede conocer la entrada.
- La salida tiene una longitud fija (como 64 bits, por ejemplo).
- Un pequeño cambio a la entrada altera mucho la salida.



USOS DE FUNCIONES HASH

- Comprobación integridad (checksum).
- Guardado de contraseña
- Huellas (comparar documentos, detección de virus, etc.)
- Firma digital



The screenshot shows a web browser window with the address bar displaying "releases.ubuntu.com/16.04/MD5SUMS". The page content lists MD5 checksums for various Ubuntu 16.04 release images. The text is as follows:

```
c94d54942a2954cf852884d656224186 *ubuntu-16.04-desktop-amd64.iso
610c4a399df39a78866f9236b8c658da *ubuntu-16.04-desktop-i386.iso
23e97cd5d4145d4105fbf29878534049 *ubuntu-16.04-server-amd64.img
23e97cd5d4145d4105fbf29878534049 *ubuntu-16.04-server-amd64.iso
494c03028524dff2de5c41a800674692 *ubuntu-16.04-server-i386.img
494c03028524dff2de5c41a800674692 *ubuntu-16.04-server-i386.iso
17643c29e3c4609818f26becf76d29a3 *ubuntu-16.04.1-desktop-amd64.iso
9e4e30c37c99b4e029b4bfc2ee93eec2 *ubuntu-16.04.1-desktop-i386.iso
d2d939ca0e65816790375f6826e4032f *ubuntu-16.04.1-server-amd64.img
d2d939ca0e65816790375f6826e4032f *ubuntu-16.04.1-server-amd64.iso
455206c599c25d6a576ba23ca906741a *ubuntu-16.04.1-server-i386.img
455206c599c25d6a576ba23ca906741a *ubuntu-16.04.1-server-i386.iso
```

ALGORITMOS DE HASH

- **Ejemplos**
- MD5,
- SHA-2, SHA-3
- **Ataques y problemas**
- Dos entradas diferentes pueden generar la misma salida (a esto lo llamamos [colisión](#)).
- La gente crea sus diccionarios metiendo en la función hash las contraseñas más usadas y obtiene una lista con sus [hashes](#).

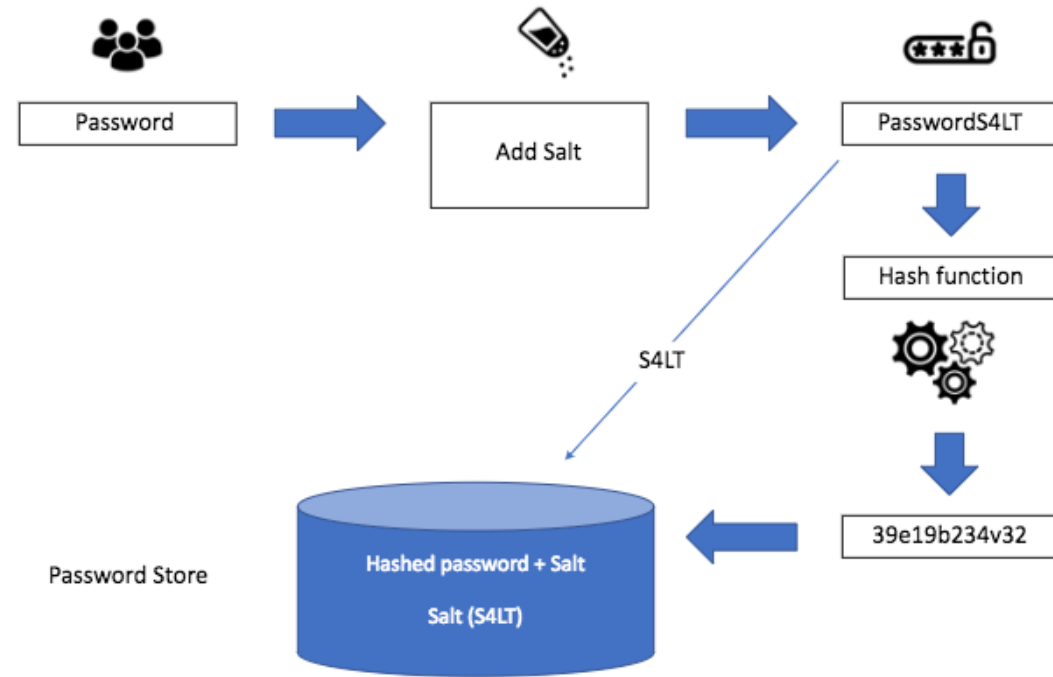


¿PERO CÓMO LO HAGO CON JAVA?

1. La clase `MessageDigest` permite realizar un hash con diferentes algoritmos, como SHA-256
2. Para los identificadores a utilizar ver [Java Security Standard Algorithm Names](#)
 - Ver `ShaHashing.java`

SALTEADO

Solución: “saltear” las contraseñas. Añadir caracteres adicionales por nuestra cuenta.



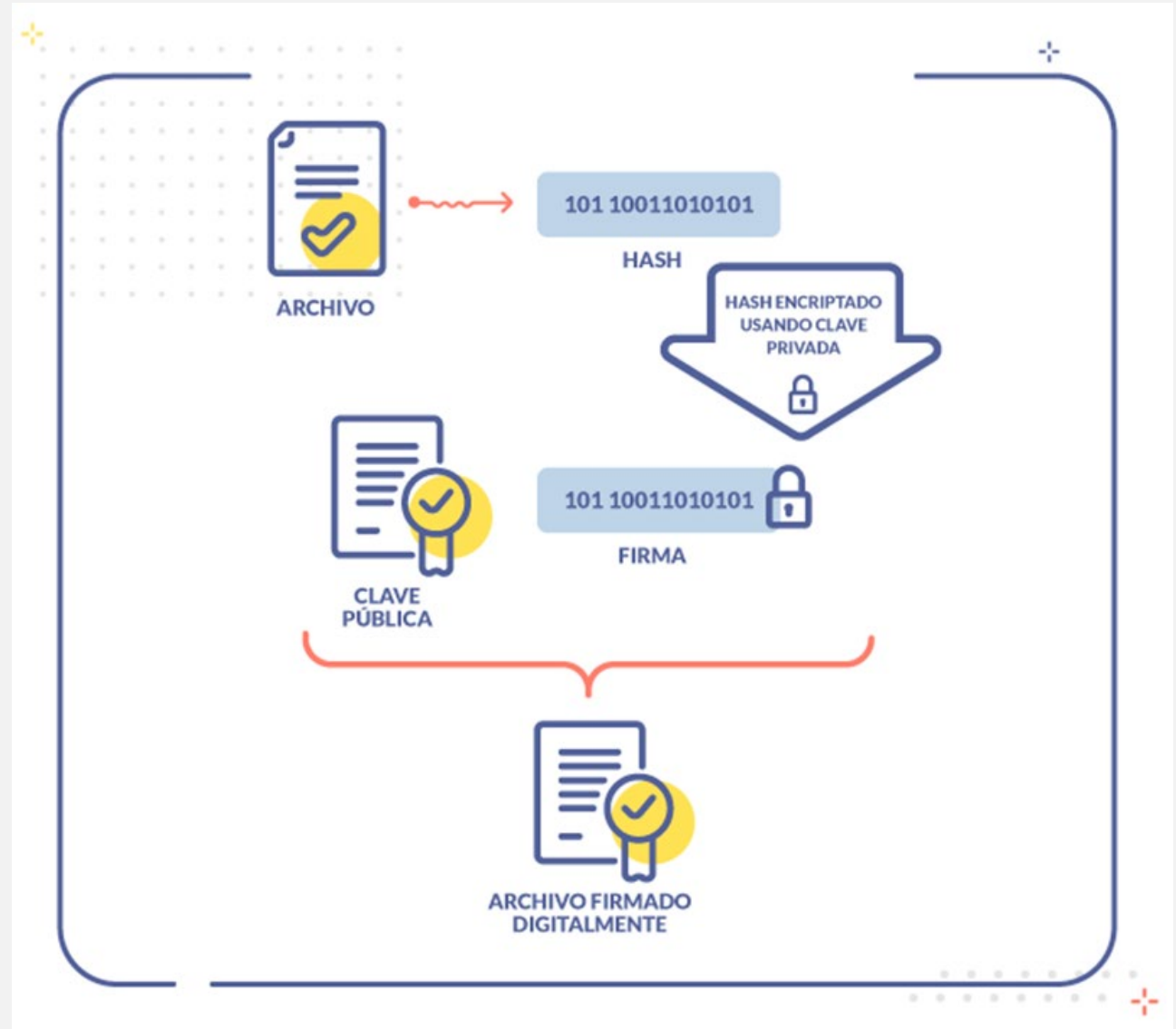
¿PERO CÓMO LO HAGO CON JAVA?

1. Reutilizamos el concepto de la generación de una clave a partir de una contraseña añadiendo una sal que vimos para AES.
 2. Para cada contraseña se utiliza una sal aleatoria distinta.
 3. Utilizamos el algoritmo [PBKDF2](#)
 4. Para los identificadores a utilizar ver [Java Security Standard Algorithm Names](#)
- Ver PasswordHashing.java

FIRMA DIGITAL

FIRMADO

- Computar el hash del archivo original
- Cifrar el hash con la clave privada -> Firma o Message Authentication Code (MAC)
- Archivo firmado compuesto de archivo original y firma



VERIFICACIÓN



VALIDACIÓN

- El receptor computa el hash del archivo original
- Con la clave pública descifra el hash del documento firmado
- Si ambos coinciden, el documento firmado es válido

¿PERO CÓMO LO HAGO CON JAVA?

1. Utilizamos la clase `Signature` que permite firmar con una clave privada
 2. Y verificar con una clave pública
 3. Para los identificadores a utilizar ver [Java Security Standard Algorithm Names](#)
 4. Por ejemplo usamos `SHA256withRSA`
 1. Hash con SHA256
 2. Cifrado con RSA
- Ver `Signing.java`

CERTIFICADOS DIGITALES



¿POR QUÉ?

Problemas

- A la hora de firmar necesitamos asegurarnos de que la clave privada del usuario que firma SOLO la tiene él.
- Necesitamos distinguir de manera fácil DE QUIÉN es cada clave pública.

SOLUCIÓN

que un tercero de confianza
firme digitalmente la clave
pública (y datos asociados a la
misma)



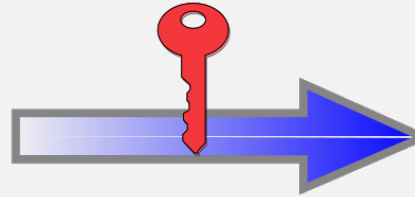
Real Casa de la Moneda
Fábrica Nacional
de Moneda y Timbre

CERTIFICACIÓN

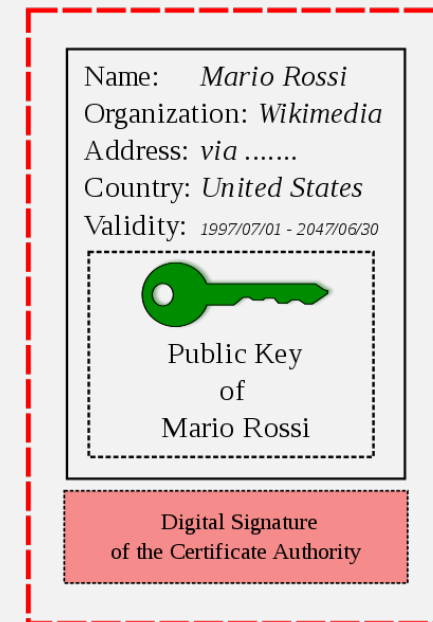
Identity Information and
Public Key of Mario Rossi



Certificate Authority
verifies the identity of Mario Rossi
and encrypts with its Private Key



Certificate of Mario Rossi

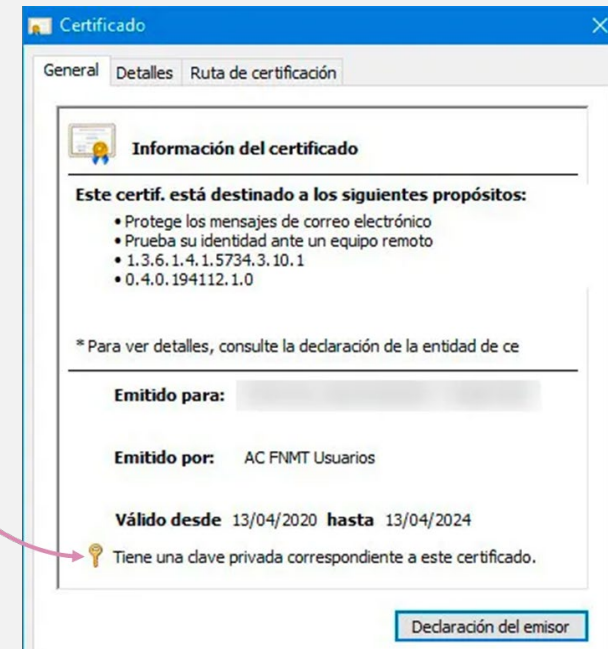


Digitally Signed by
Certificate Authority

CERTIFICADO DIGITAL

Suele contener

- La clave pública del usuario.
- Opcionalmente la privada.
- Datos (DNI, nombre, email, etc.)
- Quién emite el certificado.
- Validez.
- Formato habitual: X.509



FORMATO X.509

Certificate:

Data:

Version: 1 (0x0)
Serial Number: 7829 (0x1e95)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
68:9f

FORMATO X.509

Parte de datos

La firma de
la parte de
datos

Certificate:

Data:

Version: 1 (0x0)
Serial Number: 7829 (0x1e95)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
68:9f

FORMATO X.509

Parte de datos

La firma de
la parte de
datos

Certificate:

Data:

Version: 1 (0x0)

Serial Number: 7829 (0x1e95)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,

OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
68:9f

Quién
firma

FORMATO X.509

Parte de datos

La firma de
la parte de
datos

Certificate:

Data:

Version: 1 (0x0)

Serial Number: 7829 (0x1e95)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
68:9f

Quién
firma

Validez

FORMATO X.509

Parte de datos

La firma de
la parte de
datos

Certificate:

Data:

Version: 1 (0x0)

Serial Number: 7829 (0x1e95)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
68:9f

Quién

firma
Validez

Quién soy

FORMATO X.509

Certificate:

Data:

Version: 1 (0x0)

Serial Number: 7829 (0x1e95)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
33:35:19:d5:0c:64:b9:3d:42:b2:96:fa:f3:31:e1:
66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
68:9f

Parte de datos

La firma de
la parte de
datos

Quién
firma
Validez

Quién soy

Mi clave
pública

¿Y SI ME LO FIRMO YO?

- Certificado autofirmado:
- El propietario del certificado se lo firma él mismo con su clave privada.

Certificate:

Data:

Version: 3 (0x2)

Serial Number: 1 (0x1)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com

Validity

Not Before: Aug 1 00:00:00 1996 GMT

Not After : Dec 31 23:59:59 2020 GMT

Subject: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com

Iguales

¿QUÉ SE HACE CON ÉL?

- Se extrae la clave (o claves) que contiene y se utiliza de manera normal, como hemos visto hasta ahora.



Certificate:

Data:

Version: 1 (0x0)
Serial Number: 7829 (0x1e95)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

~~Modulus (1024 bit):~~

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
e8:35:1c:9e:27:52:7e:41:8f

~~Exponent: 65537 (0x10001)~~

Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
68:9f

VERIFICACIÓN DEL CERTIFICADO

- 1. Hago el hash a la parte de "Data".
- 2. Aplico la clave pública del emisor a la firma.
- 3. ¿Son iguales?
- La clave pública del emisor la saco de otro certificado.

Certificate:

Data:

Version: 1 (0x0)
Serial Number: 7829 (0x1e95)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
OU=Certification Services Division,
CN=Thawte Server CA/Email=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
68:9f

INFRAESTRUCTURA DE CLAVE PÚBLICA

RA: AUTORIDAD DE REGISTRO

- Una RA (autoridad de registro) es el organismo que se encarga de comprobar la identidad de la persona que va a pedir un certificado.
- Por ejemplo la [FNMT](#)



CA: AUTORIDAD DE CERTIFICACIÓN

- Una CA (autoridad de certificación) es un organismo que vincula una clave con una entidad (persona, empresa, dominio web, etc.)
- Es decir:
 - Es un organismo que tiene su propio par de claves pública y privada.
 - Se dedica a firmar las claves de otros (o sea, emitir certificados).

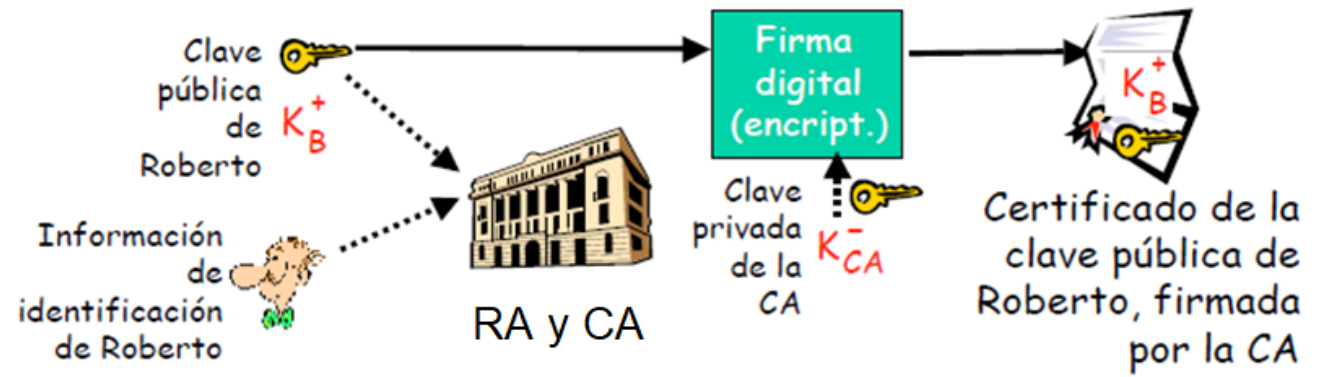
Camerfirma
Certificado Digital



Real Casa de la Moneda
Fábrica Nacional
de Moneda y Timbre

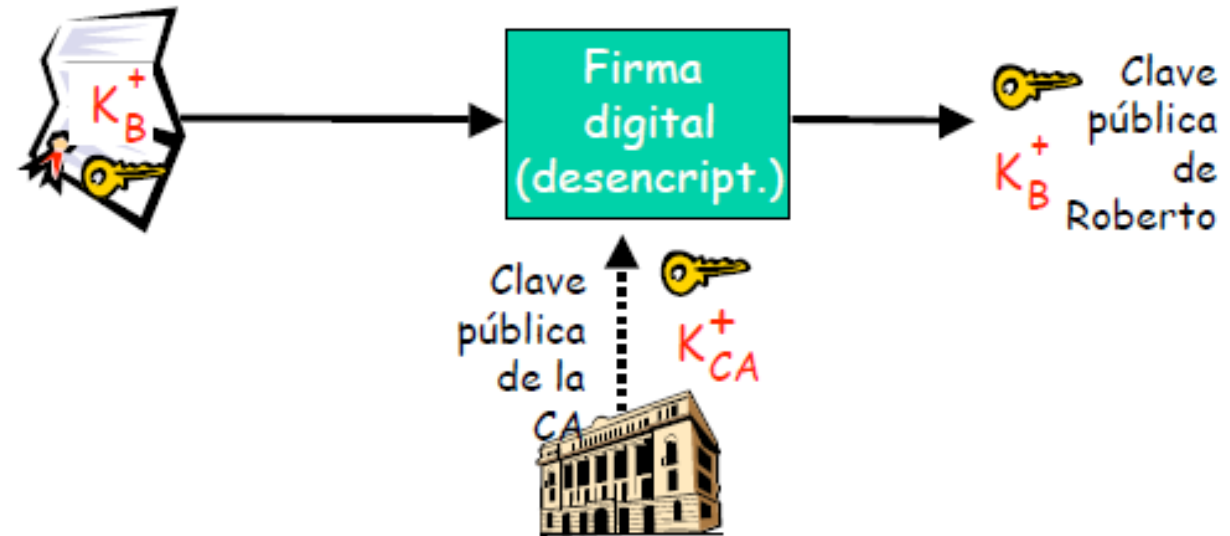
SOLICITANDO EL CERTIFICADO

- 1° Ir a una RA a verificar mi identidad.
- 2° La RA le dice a la CA que efectivamente soy yo.
- 3° La CA me emite el certificado:
 - Opción A: ya tengo un par de claves → me firma la pública.
 - Opción B: no tengo ningún par de claves (lo normal) → me las genera y me las firma.



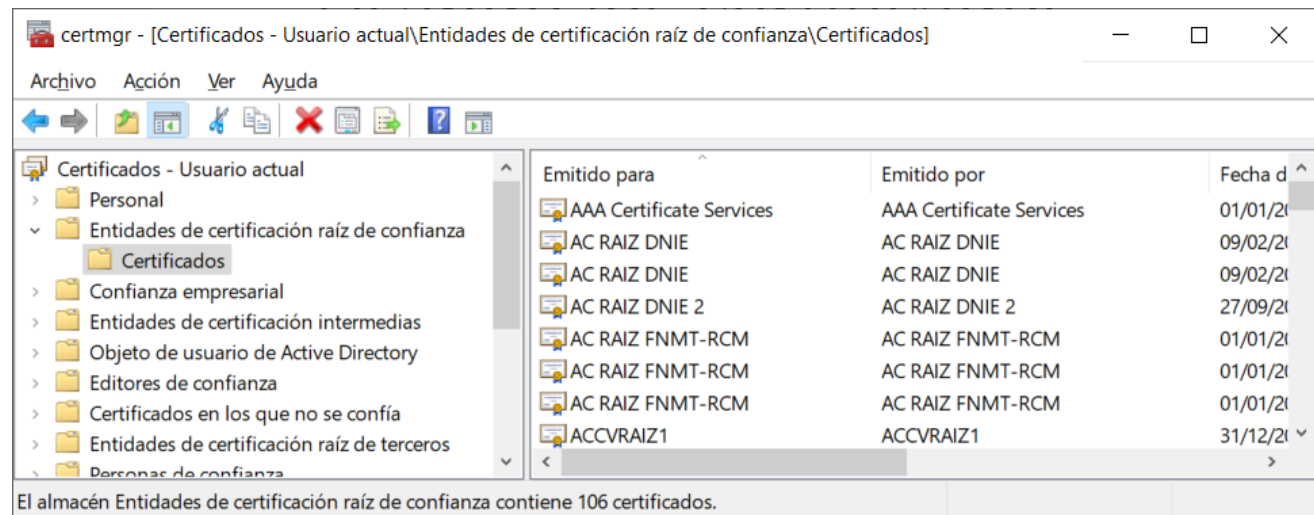
USANDO LOS CERTIFICADOS

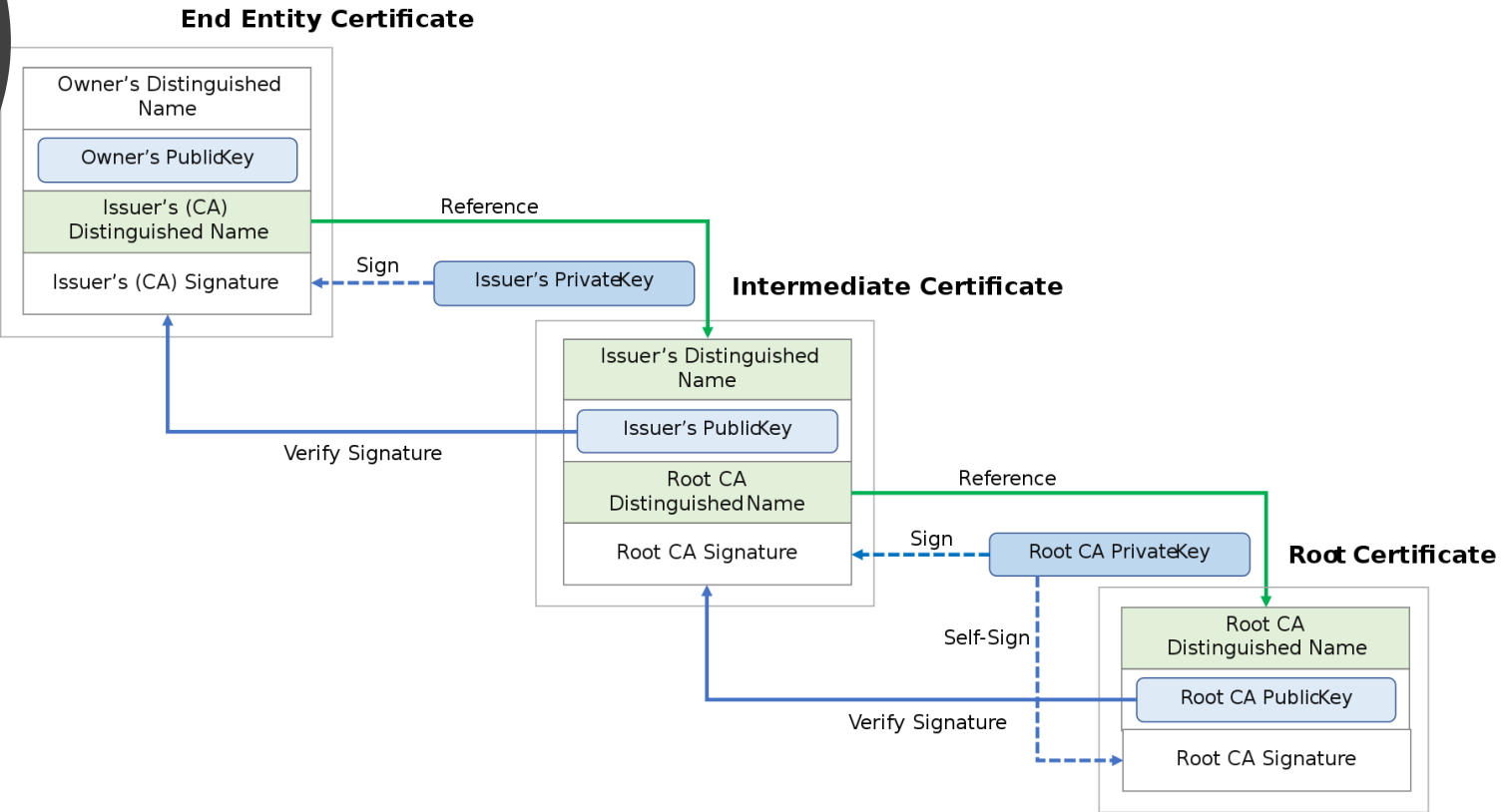
- Cuando Alicia quiere la clave pública de Roberto, ya no tiene que pedírsela a él, sino que obtiene el certificado (de Roberto, de Internet, de otra persona...) .
- Tan sólo tiene que aplicar la clave pública de la CA al certificado de Roberto para obtener la clave pública de Roberto. .



CERTIFICADOS RAÍZ

- La CA tiene su propio certificado público que se denomina certificado raíz y es autofirmado:
- El ordenador lo usa para “fiarse” de los certificados que ha generado esa CA.
 - Contiene la clave pública de la CA.
- <https://www.sede.fnmt.gob.es/descargas/certificados-raiz-de-la-fnmt>







¿POR QUÉ LOS CERTIFICADOS INTERMEDIOS?

- Certificados raíz suelen estar preinstalados en los navegadores
- Si se comprometen, esto es, se tiene acceso a su clave privada, **todos** los certificados emitidos por ellos están comprometidos
- En su lugar las CAs usan **certificados intermedios** que solo certifican un subconjunto de todos los certificados de usuario
- Mientras tanto la clave privada del certificado raíz de la CA está custodiado con [grandes medidas de seguridad](#)

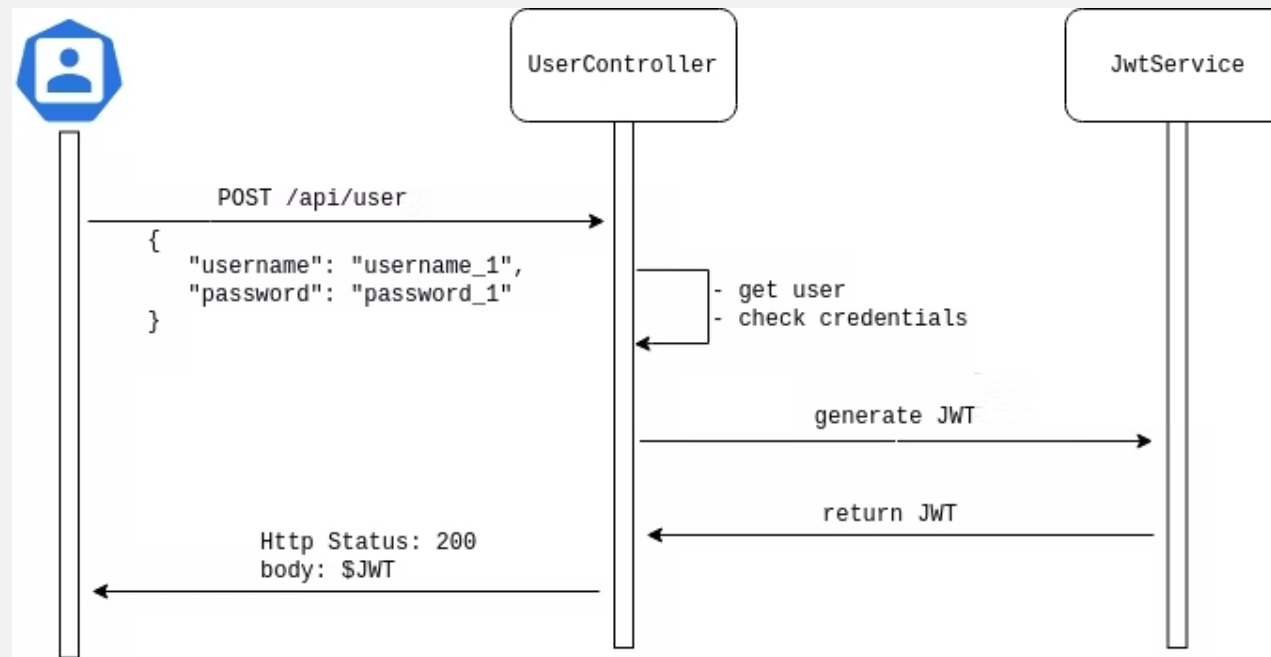
JWT

JSON Web Tokens

JSON WEB TOKENS

- [JWT](#) proporciona autenticación y autorización
- El token lo recibe el cliente cuando se autentica con su identificador y contraseña en un endpoint del servidor
- El cliente incluye el token dentro las peticiones a la API REST en la cabecera Authorization
Authorization: Bearer <token>
- El servidor verifica el token y decide si permite o no el acceso al recurso

CREACIÓN TOKEN



DEPENDENCIAS

En build.gradle:

```
implementation 'io.jsonwebtoken:jjwt-api:0.11.5'  
runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.5'  
runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.5'
```

TOKEN

- El token se compone de tres partes: cabecera, contenido y firma, separados por puntos:

xxxxxx.yyyyyy.zzzzzz

- Cada uno de los elementos se codifica en [Base64URL](#)
- Por defecto no está cifrado

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4  
gRG9lIiwiaXNTb2NpYWwiOiJmcm9udWV9.  
4pcPyMD09o1PSyXnrXCjTwXyr4BsezDI1AVTmud2fU4
```

CABECERA

- La cabecera contiene
 - Tipo de token: JWT
 - Algoritmo utilizado en la firma, por ejemplo HS256
 - Ejemplo

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

CONTENIDO

- El contenido del token es una lista de aserciones (claims) sobre una entidad (subject) que en nuestro caso es el usuario.
- Las aserciones pueden ser [registradas](#) (pertenecen al estándar), [públicas](#) (estandarizadas a posteriori) o privadas (específicas de la aplicación)

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}
```

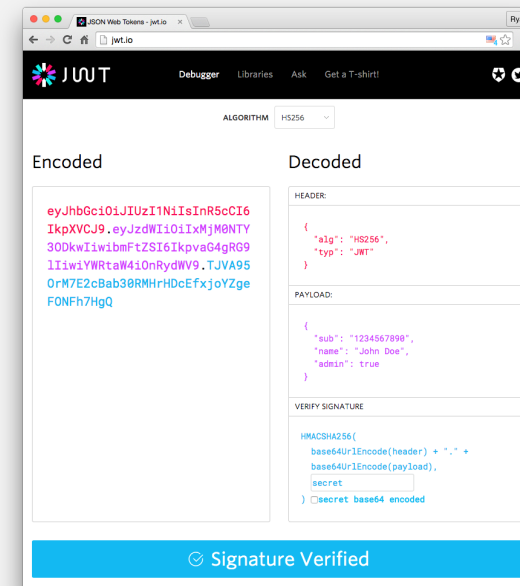
sub (registrada) identifica al usuario

name (pública) es el nombre de usuario

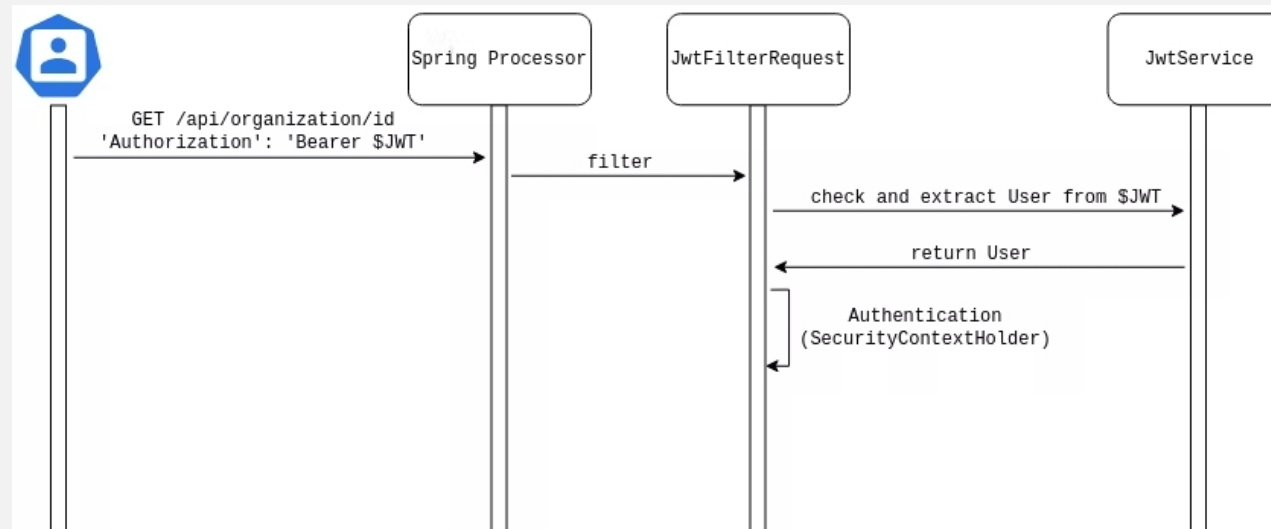
admin (privada) indica si el usuario es admin

FIRMA

- Utilizando un algoritmo de firma (recuerda, que incluye la generación de hashes y su cifrado, por lo que se requiere de una clave) se firma la parte anterior del token (cabecera.contenido)
- Al estar firmado se puede comprobar que el token no ha sido alterado
- Si la firma se realiza con una clave privada, se puede verificar al firmante
- Puedes depurar tus tokens en jwt.io



AUTENTICACIÓN



SECURITY CONFIGURATION

- `SecurityConfiguration`: establece una cadena de filtros para todas las peticiones a la API
 - Interpone un filtro para comprobar si las peticiones llevan un JWT
 - En caso de que las peticiones vayan a `/users` las deja pasar (para que se puedan autenticar)
 - El resto de peticiones no se permiten si no van acompañadas de un JWT válido

JWT REQUEST FILTER

- JwtRequestFilter se ejecuta en cada petición
- Intenta extraer un JWT de la cabecera Authorization
- Si el JWT es válido y corresponde a un usuario local, se autoriza la petición

OTRAS CLASES

- JwtService, es un servicio que permite crear y verificar JWT
- UserController, ofrece el punto de acceso para autenticar usuarios y obtener un JWT
- UserRepositoryStub, una implementación trivial con un único usuario

CÓMO PROBARLO

- Con Postman
- POST <http://localhost:8080/user> pasando un JSON en el BODY

```
{  
  "username": "elpepe",  
  "password": "pass1234"  
}
```

- Devuelve el JWT

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJlbHB1cGUlLCJpYXQiOiJlbnM0NzZcwMzEsImV4cCI6MTY3MzQ3NzYzMX0.TXKW-SImdsFEtd5NnUDqE0xrBNwQW478aoRq5maJQ5wJgFLxPL6Cx7g0a9mH07zHtBLMs5kLBaHe2gicNtCgpQ
```

CÓMO PROBARLO

- Con Postman
- GET <http://localhost:8080/teams> pasando el JWT en la cabecera Authorization:

Bearer

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJlbHB1cGUlLCJpYXQiOiJlE2NzM0NzcwMzEsImV4cCI6MTY3MzQ3NzYzMX0.TXKW-SImdsFEtd5NnUDqE0xrBNwQW478aoRq5maJQ5wJgFLxPL6Cx7g0a9mH07zHtBLMs5kLBaHe2gicNtCgpQ