Aplicaciones con comunicaciones seguras

2ºDAM IoT

Objetivos de las comunicaciones seguras

Los objetivos son:

- Confidencialidad
- Integridad
- Autenticación
- No repudio

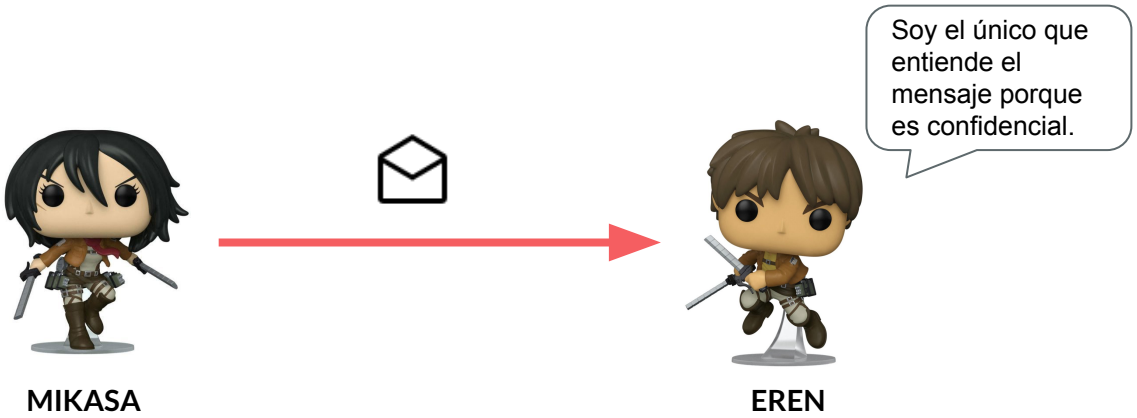
Vamos a ver todos los objetivos con un ejemplo: Mikasa envía un mensaje a Eren.

Objetivos de las comunicaciones seguras

Confidencialidad

Garantiza los datos transmitidos para que sólo estén disponibles para las entidades (personas o procesos) autorizadas a acceder a dicha información.

La información es para Eren, nadie más debe entender el mensaje.

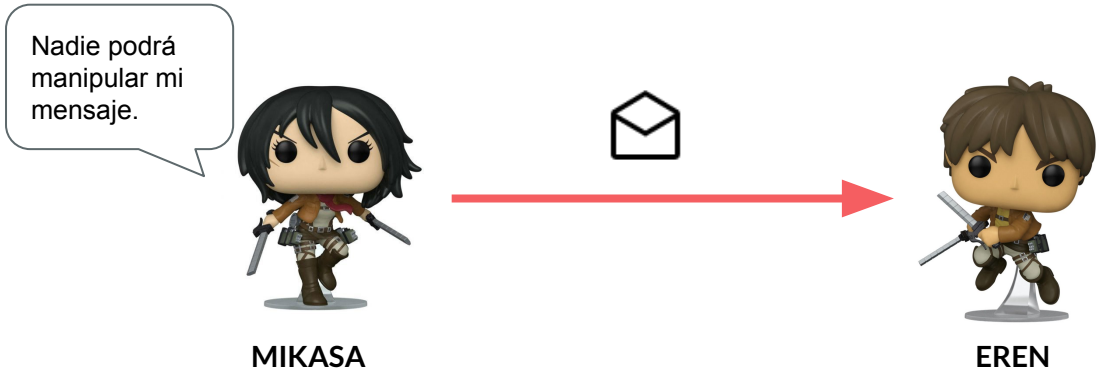


Objetivos de las comunicaciones seguras

Integridad

Garantizar los datos para que no sean modificados por alguna entidad no autorizada durante su transmisión.

Cuando Mikasa le envía información a Eren, debe asegurarse de que esa información no es alterada o manipulada por el camino

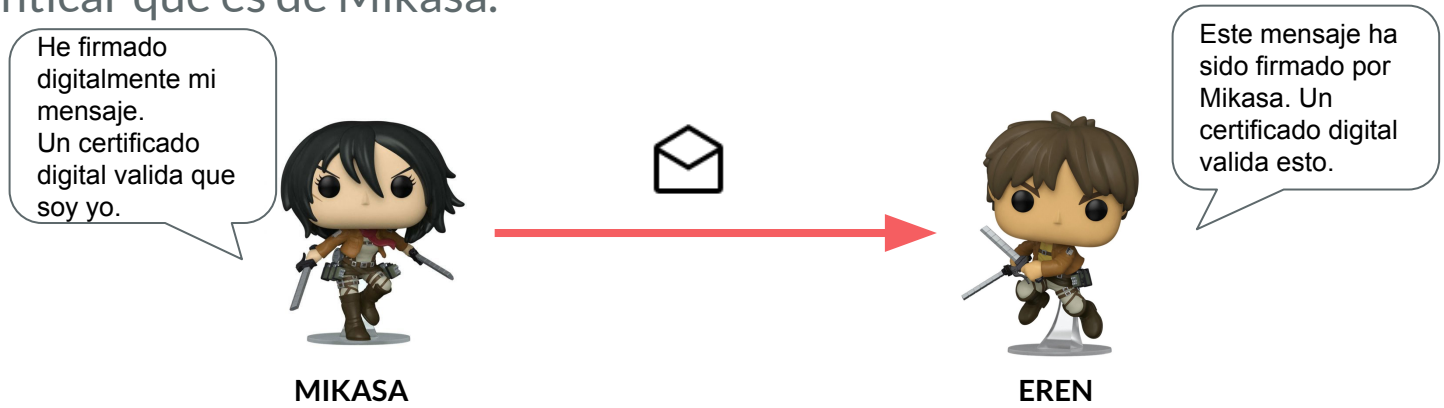


Objetivos de las comunicaciones seguras

Autenticación

Asegurar que la entidad emisora es quien dice ser. Esto nos lo proporciona la firma digital. Un certificado digital autentica la firma.

Mikasa firma digitalmente el mensaje y Eren puede comprobar la firma para autenticar que es de Mikasa.

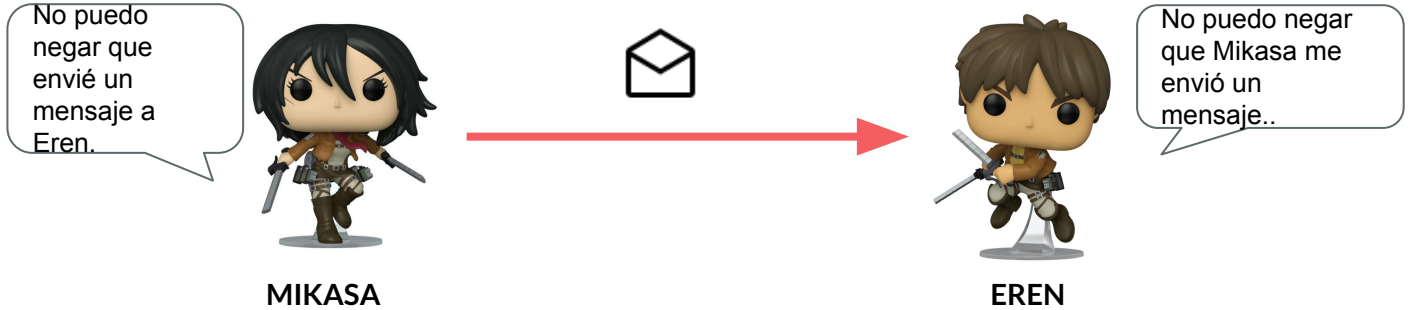


Objetivos de las comunicaciones seguras

No repudio

Garantiza la participación de las partes en una comunicación. Lo garantiza la firma digital también.

- No repudio en origen: garantiza que la persona que envía el mensaje no puede negar que es el emisor del mismo, ya que el receptor tendrá pruebas del envío.
- No repudio en destino: el receptor no puede negar que recibió el mensaje, porque el emisor tiene pruebas de la recepción del mismo.



Criptografía

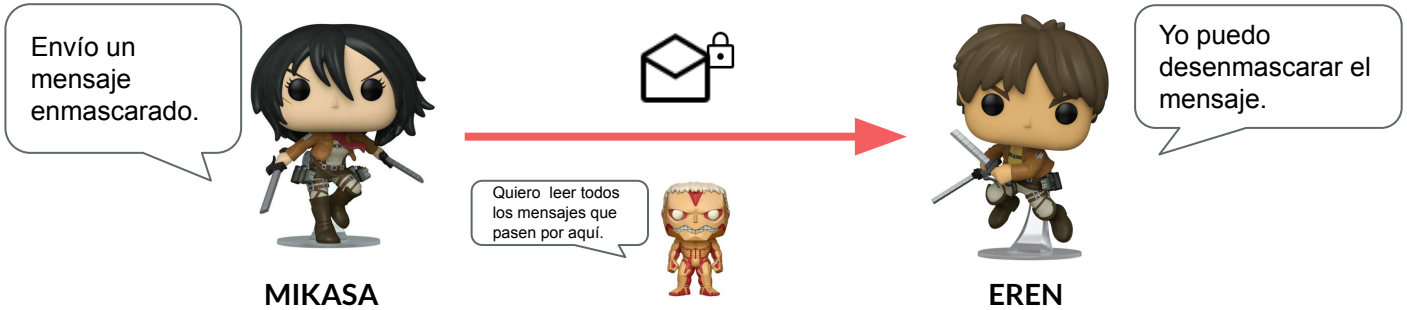
Criptografía

Los objetivos anteriores los cumplimos gracias a la criptografía.

Criptografía viene del griego, significa escritura secreta.

A lo largo de la historia se ha utilizado para:

- Enmascarar un mensaje.
- Enviar un mensaje por una línea insegura.
- Asegurar que el receptor puede leer el mensaie enmascarado.



Criptografía: Conceptos

Encriptación o cifrado de la información: Es el proceso por el cual la información o los datos a proteger son traducidos o codificados como algo que parece aleatorio y que no tiene ningún significado (datos encriptados).

Desencriptación o cifrado de la información: Es el proceso inverso, en el cual los datos encriptados son convertidos nuevamente a su forma original.

Texto llano o claro: es la información original, la que no está cifrada o encriptada.

Criptograma o texto cifrado: es la información obtenida tras la encriptación.

Algoritmo criptográfico o algoritmo de cifrado: es un conjunto de pasos u operaciones, normalmente una función matemática, usado en los procesos de encriptación y desencriptación. Asociado al algoritmo hay una clave o llave (un número, palabra, frase, o contraseña).

Criptografía: Conceptos

Clave: Controla las operaciones del algoritmo dentro del proceso de cifrado y descifrado. Puede ser:

- **Simétrica**: Misma clave para cifrar y descifrar.
- **Asimétrica**: Claves diferentes para cifrar y descifrar.

Principio Kerckhoffs: establece que la seguridad o fortaleza del sistema criptográfico debe depender exclusivamente de mantener en secreto la clave y no de ocultar el diseño del sistema, que puede ser público y conocido. De hecho, en la actualidad, la mayoría de los algoritmos utilizados en criptografía son de dominio público.

La máxima de Shannon: El adversario o enemigo conoce el sistema.

Criptografía: Conceptos

La seguridad de un sistema criptográfico depende de dos factores:

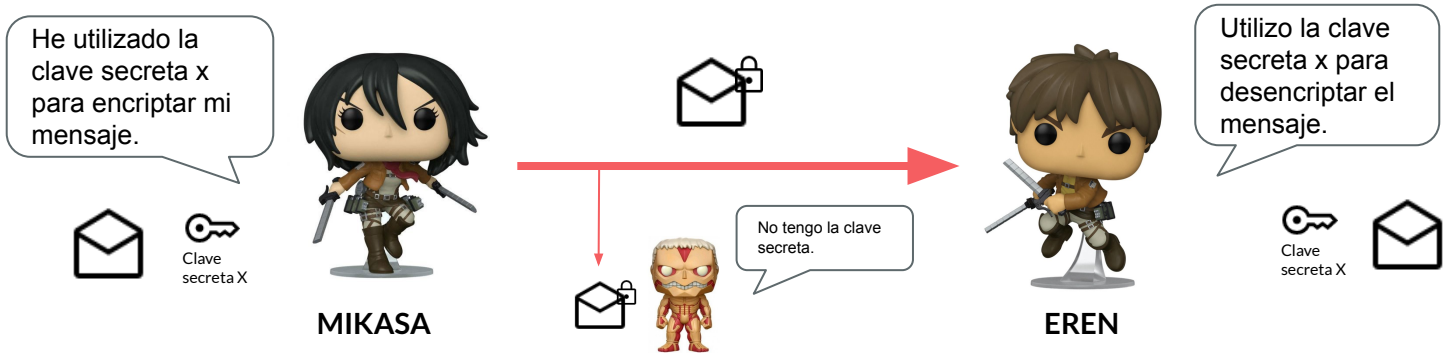
1. **El diseño o robustez del algoritmo**. Cuanto menos fallos tenga el algoritmo, más seguro será el sistema.
2. **La longitud de la clave utilizada**. A mayor longitud de la clave, mayor seguridad proporcionará el sistema. Actualmente, esta longitud se estima que debe ser como mínimo de 128 bits, existiendo algoritmos que permiten seleccionar el tamaño de la clave. Cuanto más grande sea la clave más tardo en generarla y en cifrar y descifrar.

Criptografía de clave privada o simétrica

Los algoritmos de criptografía simétrica utilizan una única **clave** para encriptar y desencriptar que se llama **secreta o privada**.

Esta clave secreta sólo la deben conocer el emisor y el receptor.

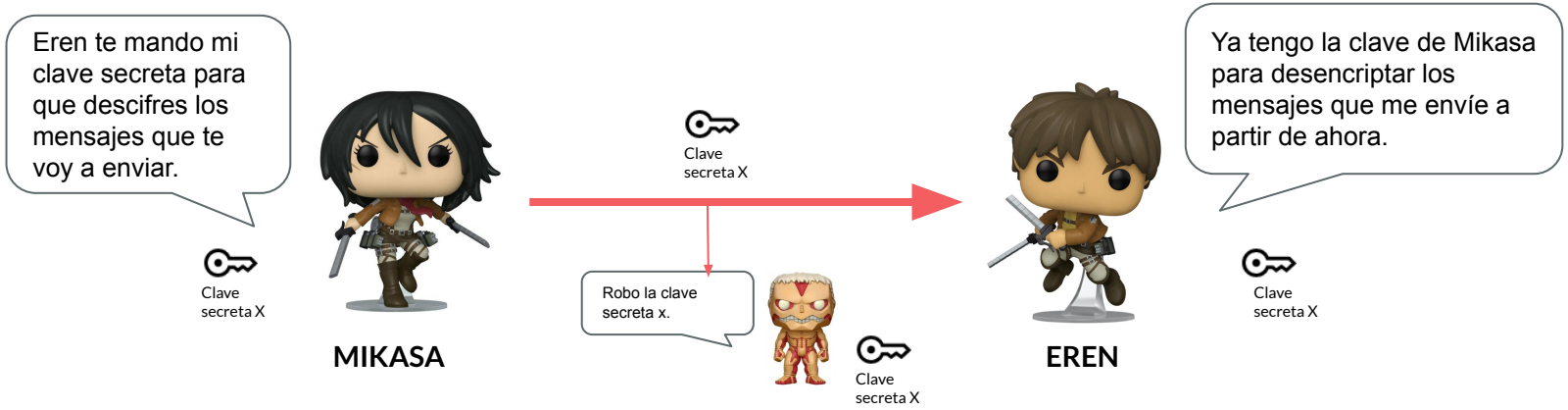
Algoritmos: **AES o Rijndael, DES, 3-DES, DES-X, IDEA y RC5**.



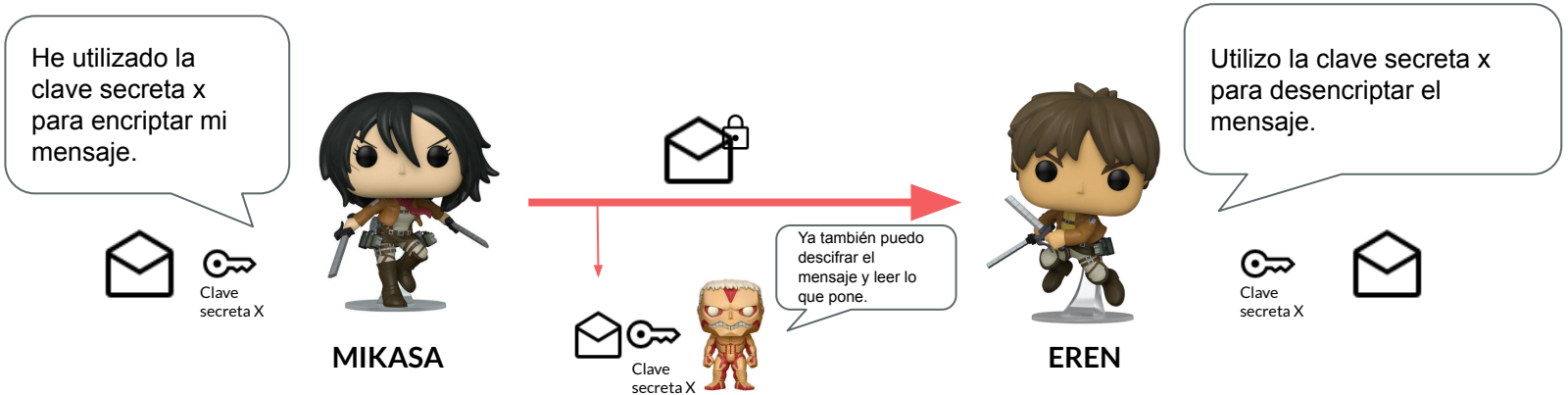
Ventajas: Son algoritmos muy rápidos. ¿Inconvenientes?

Criptografía de clave privada o simétrica

Desventajas: El receptor debe conocer la clave que se va a utilizar, lo que implica que el emisor se la debe enviar....



Criptografía de clave privada o simétrica



Otro problema es que debo generar una clave privada para todos los receptores. Pueden ser muchos.

Criptografía de clave pública o asimétrica

Surge para solucionar el problema de distribución de claves que plantea la criptografía de clave privada.

Las claves utilizadas para la encriptación y desencriptación son diferentes.

Cada parte posee una pareja de claves:

- **Pública**: conocida por todos.
- **Privada**: conocida sólo por su poseedor.

Cada pareja de claves, son complementarias: lo que cifra una de ellas, solo puede ser descifrado por su inversa.

Esa pareja de claves sólo se puede generar una vez, de modo que se puede asumir que no es posible que dos personas hayan obtenido casualmente la misma pareja de claves.

Criptografía de clave pública o asimétrica

Conocer la clave pública no permite obtener ninguna información sobre la clave privada, ni descifrar el texto que con ella se ha cifrado.

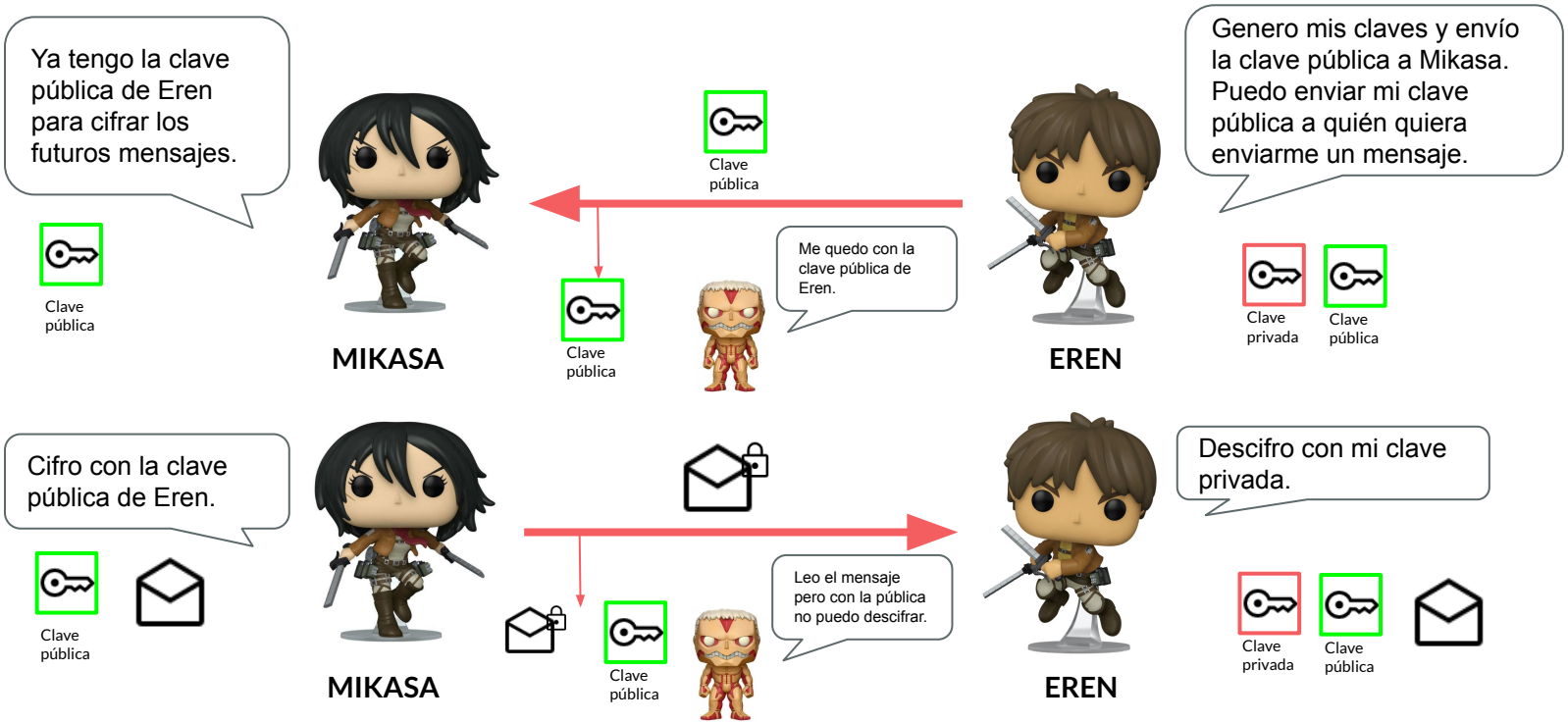
Cifrar un mensaje con la clave privada equivale a demostrar la autoría del mensaje, su autenticación. El mensaje cifrado con una clave privada podrá descifrarlo todo aquel que conozca la clave pública inversa.

Inconveniente: Los algoritmos son más lentos que los algoritmos simétricos.

Algunos algoritmos son: **DSA, RSA(estándar de facto), algoritmo de Diffie-Hellman**.

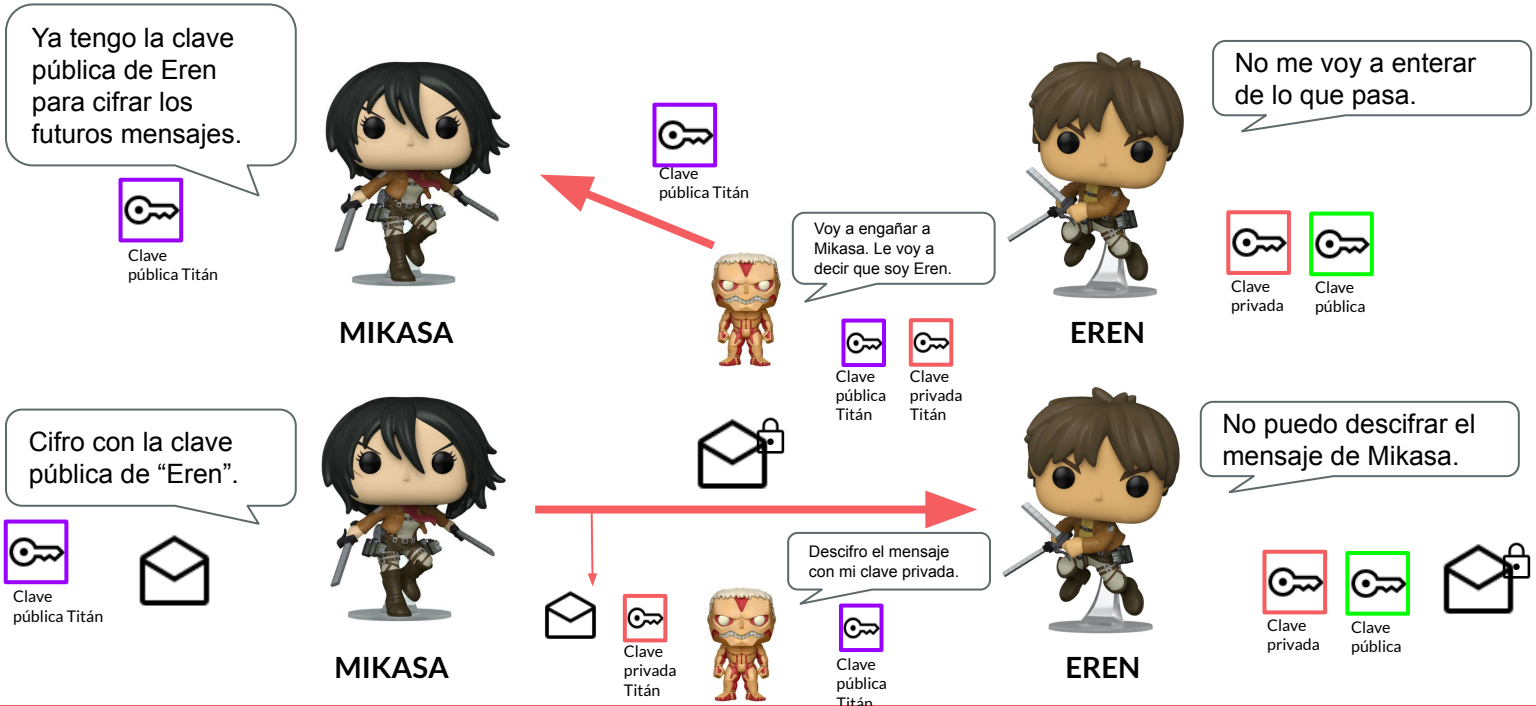
Criptografía de clave pública o asimétrica

Ciframos con la clave pública, desciframos con la clave privada.



Criptografía de clave pública o asimétrica

Man in the middle: el atacante es capaz de observar e interceptar mensajes entre las dos partes sin que ninguna de ellas sepa que el enlace entre ellos ha sido violado. Es necesario garantizar la clave pública de alguna manera, certificado digital.



Criptografía simétrica vs asimétrica

Criptografía simétrica o privada más rápida pero insegura.

Criptografía asimétrica o pública más lenta pero segura.

En la actualidad se utiliza una combinación de ambas.

Criptografía asimétrica para negociar una clave privada con la que después se comunicarán los datos.

Resumen de los mensajes

Basado en funciones HASH.

Un algoritmo de resumen toma como entrada un mensaje de longitud variable y lo convierte en un resumen de longitud fija, cuyas principales características son: siempre debe proporcionar la misma salida, debe ser aleatorio y unidireccional.

Los algoritmos HASH o de resumen más usados son **MD5 y SHA**.

Esto es muy utilizado para guardar claves de usuario en bases de datos.

- Se “hashea” una password de usuario y se guarda.
- Cuando el usuario quiere hacer login, se “hashea” el password introducido y se compara con la guardada.
- De esta manera, nadie puede ver las passwords guardadas en base de datos ni tampoco desencriptarlas.

Firmas digitales

Son el equivalente digital de las firmas personales. Se basan en criptografía de clave pública y resumen de mensajes o funciones HASH.

Para la transmisión de un mensaje entre un emisor y un receptor, el emisor transmitirá, junto con el texto deseado, la firma digital del mensaje cuya finalidad es comprobar la integridad del mensaje y la autenticidad del emisor.

El emisor debe disponer de una clave pública y otra privada.

El emisor codificará el mensaje con una función HASH cuya salida la cifrará con su clave privada, generando así la firma digital que transmitirá al receptor junto con el texto deseado.

Firmas digitales

El receptor, separará el mensaje recibido en dos partes: el texto y la firma.

Usará la clave pública del emisor para descifrar la firma y, al texto que recibe le aplicará la misma función HASH que el emisor, comparando la salida de su función con el mensaje descifrado incluido en la firma.

Si coinciden, quedará probada la integridad del mensaje y la autenticidad del emisor.

Certificados digitales

Pretenden resolver el problema de la confianza entre las partes, delegando la responsabilidad en un tercero.

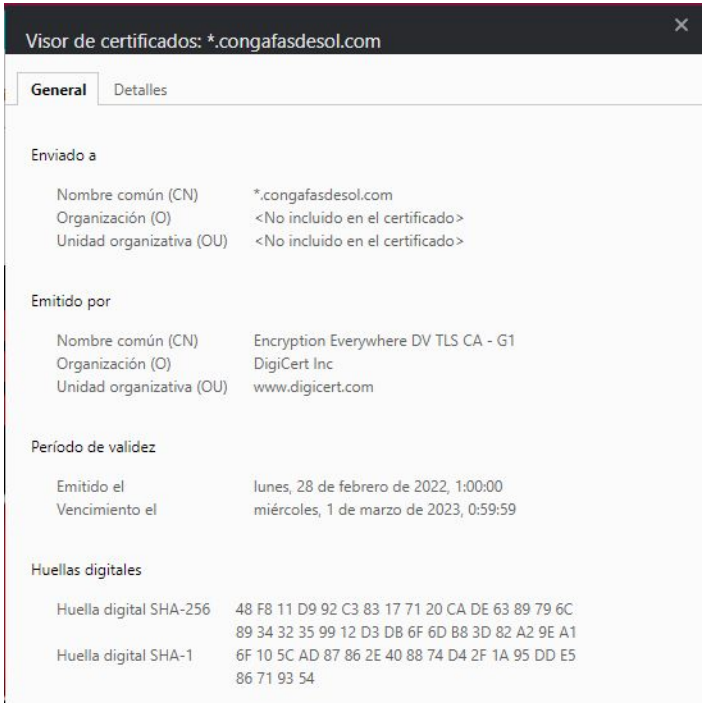
Un certificado digital no es más que un mensaje firmado por una parte de una conversación.

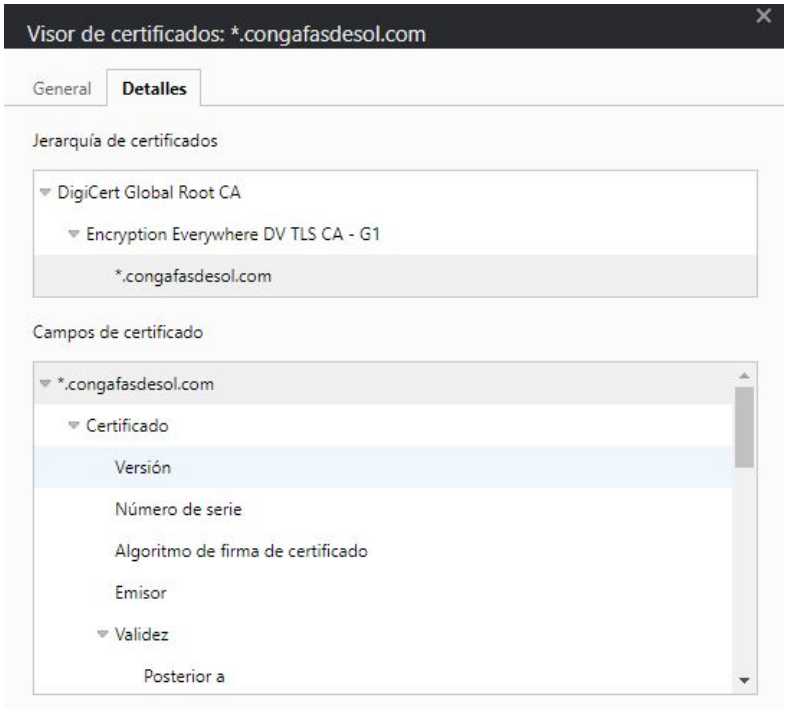
Según el **estándar X.509**, deben contener la siguiente información: número de versión, número de serie del certificado, información del algoritmo del emisor, emisor del certificado, periodo de validez, información sobre el algoritmo de clave pública, firma digital de la autoridad emisora, etc.

Entidad Certificadoras: Son organizaciones que se responsabilizan de la validez de los certificados, teniendo que, además de crearlos, proporcionar un mecanismo que permita su revocación, su suspensión, la búsqueda de certificados y la comprobación del estado del certificado.

Certificados digitales

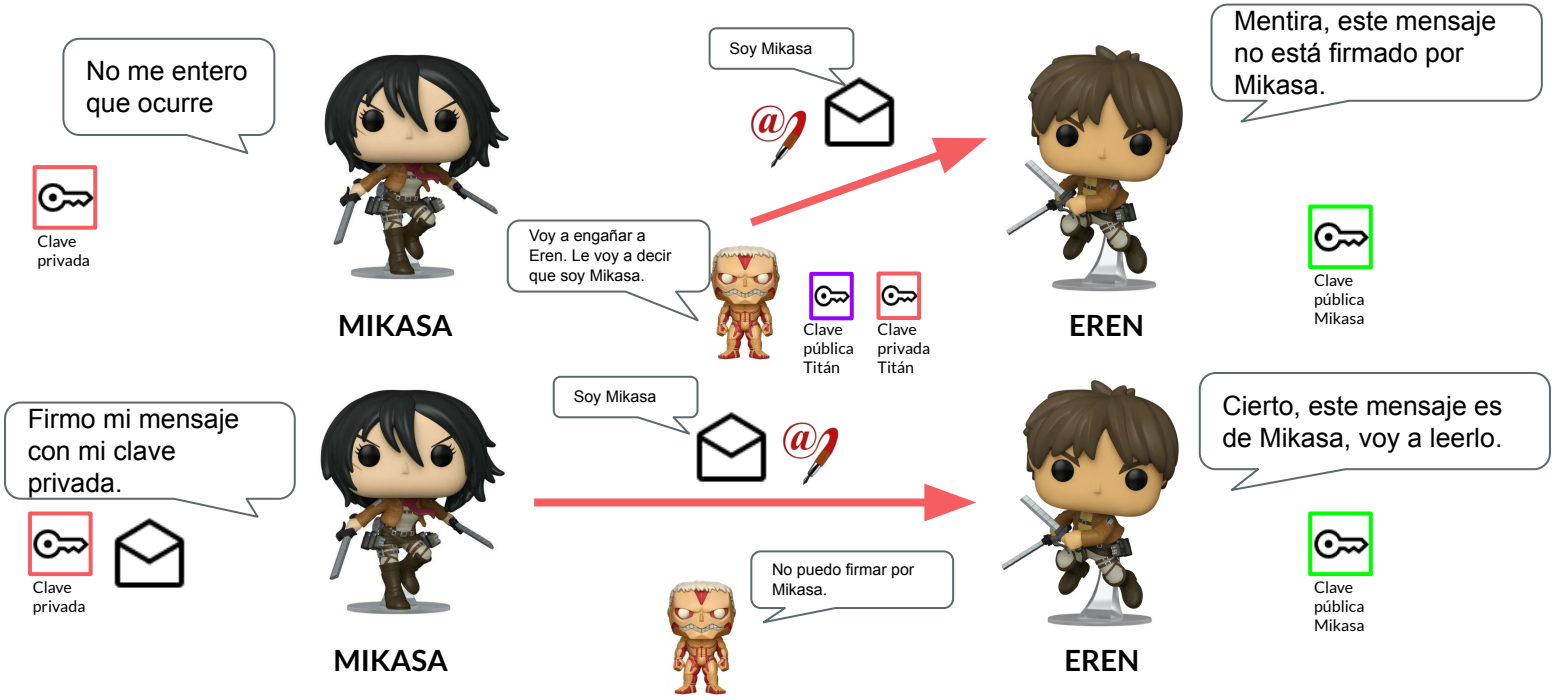
Ejemplo de un certificado digital del dominio congafasdesol.com





Firmas digitales

Para la firma digital, se realiza un hash del mensaje, se cifra con la privada para validar la autenticidad y se descifra con la pública. La clave pública suele estar publicada en un repositorio público para validar firmas.



Criptografía en Java

Criptografía en Java

Un proveedor de servicios criptográficos no es ni más ni menos que una empresa de seguridad que genera sus propios servicios de seguridad en Java.

Las clases más importantes de criptografía en Java son:

- **MessageDigest**: Para resúmenes de mensajes.
- **Signature**: Para firmas digitales.
- **KeyFactory**: Factoría para el manejo de claves seguras.
- **KeyPairGenerator**: Para generar claves públicas y privadas.
- **KeyGenerator**: Para generar claves secretas.
- **Cipher**: Para encriptación y desencriptación de información.

Cada una de estas clases incluyen un método **getInstance()** que devuelve un algoritmo criptográfico de un proveedor dado.

Gestión de claves

En la generación de claves se utilizan números aleatorios seguros, que son números aleatorios que se generan en base a una semilla.

El paquete **java.security** proporciona las siguientes clases para la gestión de claves:

- El interface **Key**, permite la representación de claves, su almacenamiento y envío de forma serializada dentro de un sistema.
  - **getAlgorithm()**: Devuelve el nombre del algoritmo con el que se ha generado la clave (RSA, DES, etc.).
  - **getEncoded()**: Devuelve la clave como un array de bytes.
  - **getFormat()**: Devuelve el formato con el que está codificada la clave

Gestión de claves

- La clase **KeyPairGenerator** permite la generación de claves públicas y privadas (asimétricas). Genera objetos del tipo **KeyPair**, que a su vez contienen un objeto del tipo **PublicKey** y otro del tipo **PrivateKey**.
  - **initialize()** permite establecer el tamaño de la clave y el número aleatorio a partir del cual será generada.
  - **generateKeyPair() / genKeyPair()** devuelve un objeto KeyPair con la PublicKey y la PrivateKey.
- La clase **KeyGenerator** permite la generación de claves secreta (simétricas). Genera objetos de tipo **SecretKey**.
  - **init()** permite establecer el tamaño de la clave y el número aleatorio a partir del cual será generada.
  - **generateKey()** devuelve un objeto SecretKey con la clave privada/secreta.

Gestión de claves

- La clase **SecureRandom** permite generar números aleatorios seguros.
  - El método **setSeed()** permite establecer el valor de la semilla.
  - El constructor **secureRandom()** utiliza la semilla del proveedor SUN.
  - El método **next()** y el **nextBytes()** obtienen el valor de los números generados.

La creación de claves se basa en el tamaño de las mismas, de manera que, si incrementas mucho el tamaño de una clave, el tiempo de cálculo de la misma también se incrementará; y esto puede suponer que la administración de claves no sea lo suficientemente ágil para una determinada aplicación.

Ejemplo AulaVirtual → GeneraClaves

MessageDigest

La clase **MessageDigest** del **paquete java.security** permite la creación de resúmenes de mensajes con el algoritmo y proveedor especificados. Los métodos que debes utilizar para crear un resumen de mensaje son:

- **getInstance()**: obtiene el algoritmo de resumen.
- **update()**: obtiene el resumen.
- **digest()**: completa la obtención del resumen.

En JDK podemos encontrar dos algoritmos de resumen de mensajes:

- **MD5**: Genera una salida de 128 bits de longitud fija.
- **SHA-1**: Genera una salida de 160 bits.

Ejemplo Aula Virtual → EjemploMessageDigest

MessageDigest

```
MessageDigest md = MessageDigest.getInstance( algorithm: "SHA1");
String texto = "Texto para el mensaje ejemplo SHA1";
md.update(texto.getBytes());
byte[] resumen = md.digest();
```

Para imprimir los bytes por pantalla puedes utilizar los siguiente:

Base64.Encoder encoder = Base64.getEncoder();

String cadena = encoder.encodeToString(<<*array de bytes*>>);

Signature

Permite realizar una firma digital, así como hacer su verificación.

Pasos a seguir para realizar la firma de un mensaje y verificarla:

- Generar las claves públicas y privadas mediante la clase **KeyPairGenerator**:
  - La PrivateKey la utilizaremos para firmar.
  - La PublicKey la utilizaremos para verificar la firma. Realizar la firma digital mediante la clase Signature y un algoritmo asimétrico, por ejemplo DSA.
- Crearemos un objeto Signature.
  - Al método **initSign()** le pasamos la clave privada.
  - El método **update()** creará el resumen de mensaje.
  - El método **sign()** devolverá la firma digital.

Signature

Firma de un array de bytes

- Primero debemos pasar los datos a bytes.
- Firmo con la clave privada.
- sign() devuelve un array de bytes correspondientes con la firma.

```
//crea el objeto tipo Signature con algoritmo DSA
Signature firma = Signature.getInstance( algorithm: "DSA");
//inicializa la firma con la clave privada a utilizar
firma.initSign(clave);
//obtine el resumen del mensaje
firma.update(datos);
//obtien la firma digital
firmado = firma.sign();
```

Signature

- Verificar la firma creada mediante la clave pública generada.
  - Al método **initVerify()** le pasaremos la clave pública.
  - Con **update()** se actualiza el resumen del mensaje para comprobar si coincide con el enviado.
  - El método **verify()** realizará la verificación de la firma.

Signature

Verificación de una firma

- Verificamos la firma con la clave pública.
- Pasamos el texto original en bytes.
- Comprobamos con el texto firmado.

```
//crea el objeto tipo Signature con algoritmo DSA
Signature firma = Signature.getInstance( algorithm: "DSA");
//verifica la clave pública
firma.initVerify(clave);
//actualiza el resumen de mensaje original
firma.update(texto);
//devuelve el resultado de la verificación
boolean firmaCorrecta = firma.verify(textoFirmado);
```

Cipher

Permite realizar encriptación y desencriptación, tanto con clave pública como privada (secreta).

- **getInstance()**: se indica el algoritmo y proveedor que utilizará el objeto cifrador Cipher.
- **init()**: se indicará el modo de operación del objeto Cipher:
  - **Cipher.ENCRYPT_MODE**: Es el modo encriptación.
  - **Cipher.DECRYPT_MODE**: Es el modo desencriptación.
- **update()** y **doFinal()** se insertarán los datos en el cifrador.

Cipher

La encriptación mediante un objeto Cipher puede ser:

- De bloque o Block Cipher: El texto a cifrar se divide en bloques de una tamaño fijo. Cada uno de estos bloques se cifra de manera independiente, y posteriormente se construye todo el texto cifrado. Si el texto a cifrar no es múltiplo de 64 se completa con un relleno o padding. Por ejemplo: Cipher.getInstance("Rijndael/ECB/PKCS5Padding").
- De flujo o Stream Cipher: El texto se cifra bit a bit, byte a byte o carácter a carácter, en lugar de bloques completos de bits. Resulta muy útil cuando hay que transmitir información cifrada según se va creando, es decir, se cifra sobre la marcha.

Si leemos un fichero poco a poco → update() y doFinal()

Si es una cadena → doFinal() directamente

Cipher

Ejemplos del aula virtual:

- Cifrado de un fichero con clave privada y algoritmo DES.
  - **EjemploDES**
- Cifrado de un texto con clave pública y algoritmo RSA.
  - **EjemploRSA**