Semana 14: Encriptación

Cifrado asimétrico

Aprendizajes esperados

Contenidos:

- Características y principios del cifrado asimétrico
- Algoritmos de cifrado asimétrico
- Funciones de hash

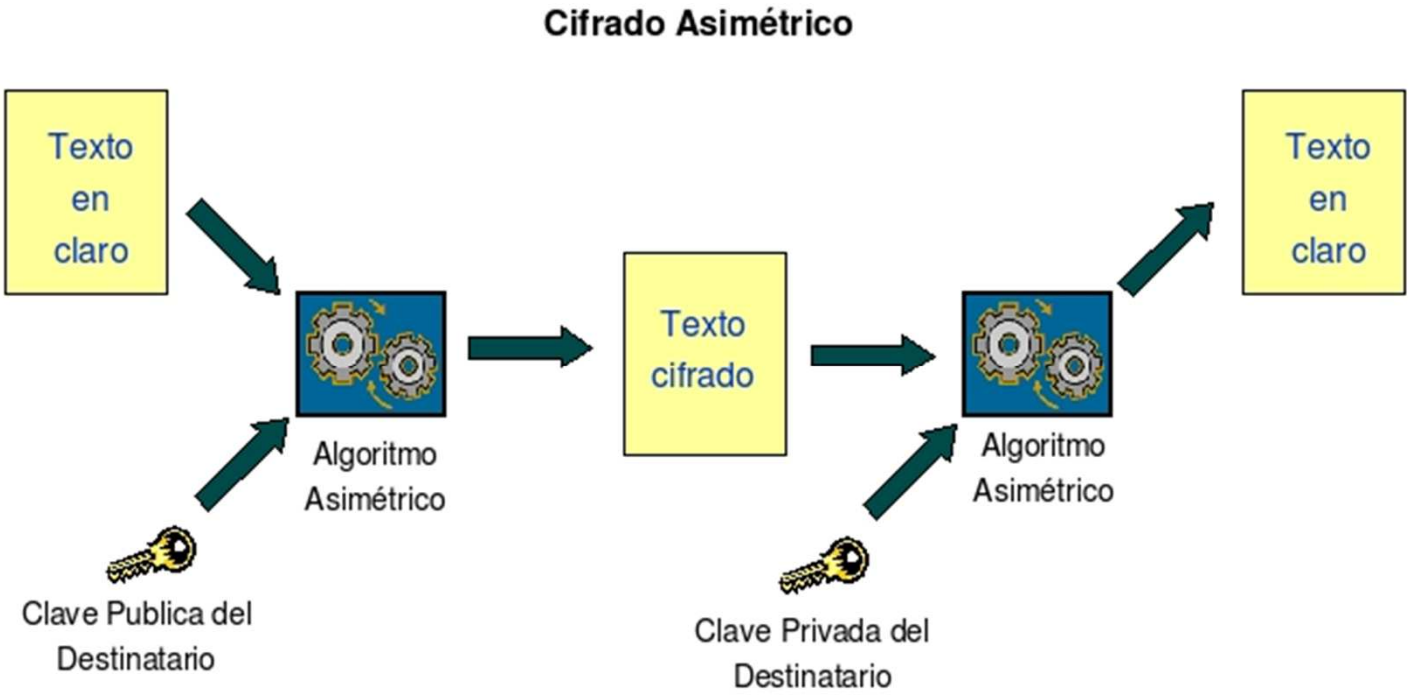
Encriptación Asimétrica

- Los **CIFRADORES ASIMÉTRICOS** o **cifradores de claves públicas** involucran una **clave pública**, que puede ser libremente distribuida, y una **clave privada**, que debe ser secreta.
- Estas claves siempre son generadas en parejas.
- Las **claves públicas** realmente son públicas, nadie puede violar tu privacidad sin la **clave privada**.

Encriptación Asimétrica

- El mecanismo de distribuir las **claves públicas**, sin embargo, es un gran reto.
- Los datos encriptados usando la **clave pública** pueden ser desencriptados usando la **clave privada**. Ninguna otra clave puede desencriptar los datos, y la **clave privada** solo puede desencriptar los datos que fueron encriptados con su pareja (la **clave pública**).

Encriptación Asimétrica



Encriptación Asimétrica

- La **clave pública** es usada para encriptar los datos y la **clave privada** se usa para desencriptar los datos.
- La **clave pública** encripta, pero NO puede ser utilizada para desencriptar. Sólo la **clave privada** puede desencriptar los datos.

Encriptación Asimétrica

- La **clave privada** no puede deducirse de la **clave pública**, por lo que no hay peligro en transmitir las claves públicas por la red.
- Algunos algoritmos de **cifrado asimétrico** son: **RSA**, **DSA**, **Diffie-Hellman**.

RSA

- Este algoritmo fue inventado por R. Rivest, A. Shamir y L. Adleman (de sus iniciales proviene el nombre del algoritmo) en el Massachusetts Institute of Technology (MIT).
- **RSA** emplea las ventajas proporcionadas por las propiedades de los **números primos** cuando se aplican sobre ellos operaciones matemáticas basadas en la función módulo.

RSA

- La robustez del algoritmo se basa en la facilidad para encontrar dos números primos grandes, frente a la enorme dificultad que presenta la factorización de su producto.
- Aunque el avance tecnológico hace que cada vez sea más rápido un posible ataque por fuerza bruta, el simple hecho de aumentar la longitud de las claves empleadas supone un incremento en la carga computacional lo suficientemente grande para que este tipo de ataque sea inviable.

RSA

- Sin embargo, se ha de notar que, aunque el hecho de aumentar la longitud de las claves **RSA** no supone ninguna dificultad tecnológica, las leyes de exportación de criptografía de Estados Unidos imponen un límite a dicha longitud.

Encriptación Asimétrica con OpenSSL

- Por ejemplo, para generar una clave privada de tipo **RSA** (por defecto de 512 bits) utilizamos la opción **genrsa**:

```
# openssl genrsa
Generating RSA private key, 512 bit long modulus
.........++++++++++++++++++++++++++
e is 65537 (0x10001)
-----BEGIN RSA PRIVATE KEY-----
MIIBOwIBAAJBALcy7h39eVFsowMCf8PEF5efb8KrC
JBtxCy
. . .
OG3DHZTvhNcpbT5xjVont0uKc9o7/22AqEnJEU7pn
O==
```

Encriptación Asimétrica con OpenSSL

- Para generar una llave privada RSA de 1024 bits:

```
# openssl genrsa 1024
```

- Para generar una llave privada RSA de 1024 bits y dejarla almacenada en privada.key:

```
# openssl genrsa -out privada.key 1024
```

Encriptación Asimétrica con OpenSSL

- La opción **rsa** de **OpenSSL** produce una versión pública (en pantalla) de la clave privada **RSA**:

```
# openssl rsa -in privada.key -pubout
```

- Para extraer una clave pública al archivo publica.key:

```
# openssl rsa -in privada.key -pubout -
  out publica.key
```

Encriptación Asimétrica con OpenSSL

- Para encriptar fichero.txt con la clave pública se utiliza la opción **rsautl** de **OpenSSL**:

```
# openssl rsautl -encrypt -pubin -inkey
  publica.key -in  fichero.txt -out
  cifrado.txt
```

- Para descifrar con la clave privada:

```
# openssl rsautl -decrypt -inkey
  privada.key  -in  cifrado.txt -out
  fich_recup.txt
```

Funciones de Hash

- Se puede definir una **FUNCIÓN HASH** como aquella que reduce el mensaje a un conjunto de datos, denominado **resumen**, de longitud mucho menor que el mensaje, usualmente 128 ó 254 bits
- Las **funciones de hash** generan un cifrado unidireccional e irreversible: lo que se cifra de esta manera no puede ser descifrado.

Funciones de Hash

- No hace falta una clave. El mensaje cifrado depende exclusivamente del original.
- El mensaje cifrado es normalmente de longitud fija y mucho más corto que cualquier mensaje típico.
- Se trata de una función libre de **colisiones** en sentido estricto (es muy difícil encontrar un par de mensajes cuyo cifrado sea equivalente).

Funciones de Hash

- Cualquier alteración del mensaje original, por pequeña que sea, genera un mensaje cifrado completamente distinto.
- Entre las funciones de hash más populares podemos encontrar: **MD2**, **MD4**, **MD5**, **SHA**, **SHA1**.

MD* - Message Digest *

- Los algoritmos de hash **MD2**, **MD4** y **MD5** fueron creados por Ron Rivest de **RSA Security** en 1989, 1990 y 1992, respectivamente.
- **MD5** es la última versión de algoritmos de hash de **RSA**, maneja optimización para procesadores de 32 bits; es uno de los algoritmos hash más difundidos, proporciona un aceptable nivel de seguridad y resulta más rápido que **SHA**.

MD* - Message Digest *

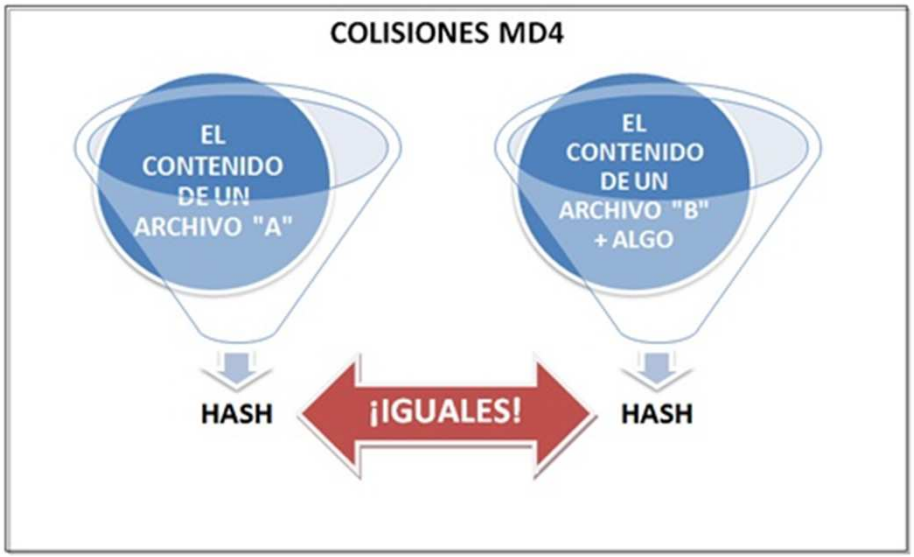
- Esta familia de algoritmos de hash puede recibir como entrada mensajes de cualquier longitud entregando valores hash de 128 bits.
- La descripción de los tres algoritmos se encuentra documentada en los **RFCs 1319**, **1320** y **1321**, respectivamente.

MD* - Message Digest *

- **MD5** es una versión mejorada de **MD4**, es ligeramente más lento pero en compensación es más seguro.
- El algoritmo maneja bloques de 512 bits, cada bloque es sometido a cuatro ciclos de cálculos.
- Se han encontrado vulnerabilidades de colisión en **MD5**.

MD* - Message Digest *

- Pero **¿qué son las colisiones?**
- Es la posibilidad de que un mismo hash de archivo pudiera ser asignado a más de dos archivos con diferente contenido.



SHA – Secure Hash Algorithm

- **SHA** es el algoritmo reconocido como estándar hash seguro por el **NIST** (**FIPS 180**); este estándar forma parte de un proyecto denominado **Capstone** y fue publicado en 1994.
- **SHA** es ligeramente más lento que **MD5** y su diseño es similar al de **MD4**.
- Presenta mayor resistencia ante ataques de fuerza bruta.

SHA – Secure Hash Algorithm

- **SHA-1** (**FIPS 180-1**) es la revisión de **SHA**, se encuentra registrado como norma **ANSI X9.30**. El algoritmo trabaja sobre mensajes de menos de 2^{64} bits entregando valores hash con una longitud de 160 bits; tiene mayor resistencia a colisiones.
- El estándar actual (**FIPS 180-2**) especifica cuatro variaciones para la representación de los datos **SHA-1**, **SHA-256**, **SHA-384** y **SHA-512**.

SHA – Secure Hash Algorithm

- Cuando un mensaje tiene una longitud menor que 2^{64} se ocupa **SHA-1** y **SHA-256**, en cambio si el tamaño del mensaje es menor a 2^{128} se ocupa **SHA-384** y **SHA-512**.

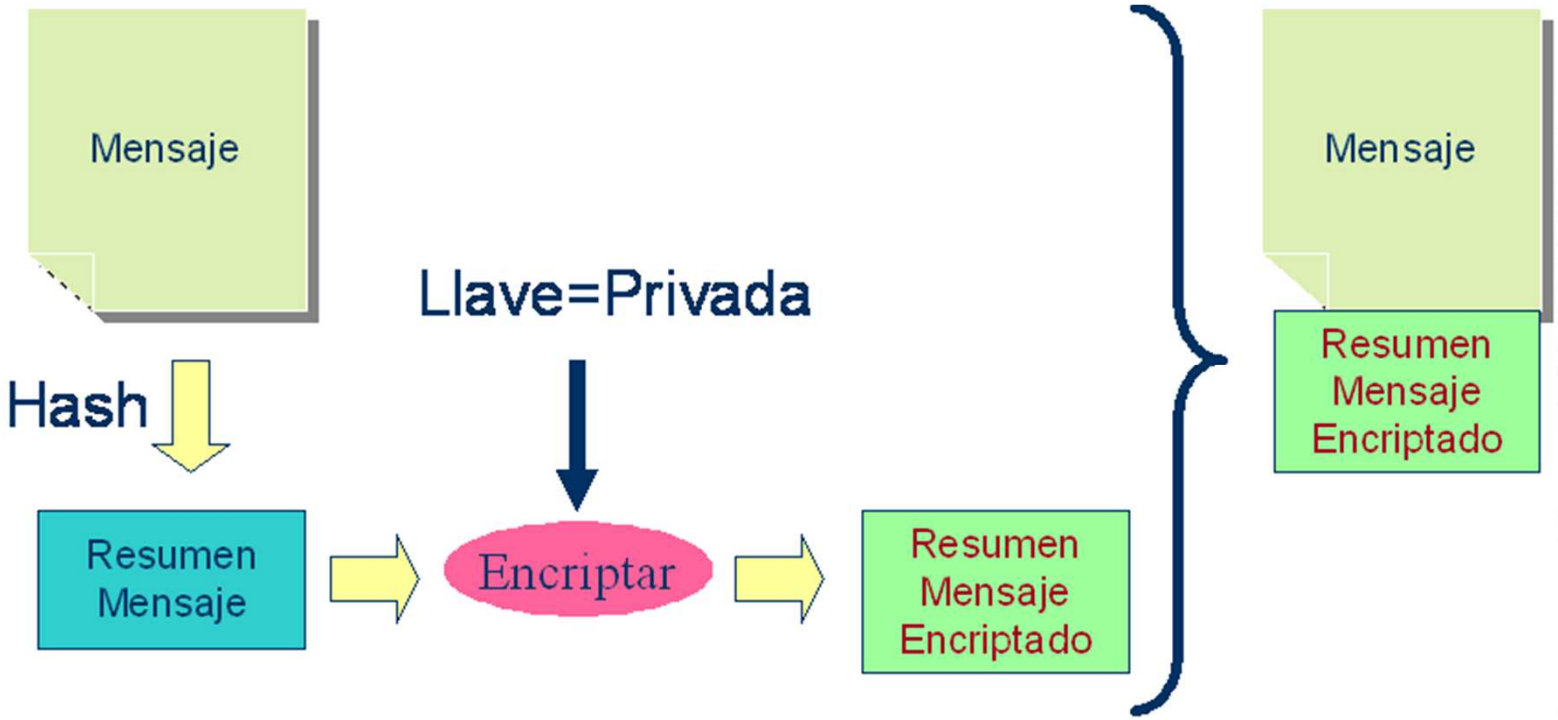
Resúmenes con OpenSSL

- Los **resúmenes** (o **digests**) de mensajes (o archivos) son usados para asegurar que un mensaje (o archivo) es válido y no ha sido modificado durante la transmisión.
- El resumen es creado aplicando una **función de hash** sobre el mensaje (o archivo) original. Es extremadamente difícil (sino imposible) encontrar dos mensajes para los cuales se obtenga un mismo valor de resumen.

Resúmenes con OpenSSL

- El mensaje y su resumen son encriptados y enviados al receptor. Después de la desencriptación, el receptor calcula el resumen del mensaje y compara dicho valor con el resumen recibido para asegurar la integridad del mensaje.
- Si un intruso modifica los datos encriptados durante la transmisión, los datos encriptados NO tendrán un resumen válido.

Resúmenes con OpenSSL



Resúmenes con OpenSSL

- La orden **list-message-digest-commands** de **OpenSSL** muestra una lista de tipos de resúmenes disponibles en la instalación local de **OpenSSL**:

```
# openssl list-message-digest-commands
```

Resúmenes con OpenSSL

- Los **resúmenes (digests)** se generan con la opción **dgst** de **OpenSSL**.
- Para obtener un resumen **MD5** de un archivo llamado fichero.txt:

```
# openssl dgst –md5 fichero.txt
```

Resúmenes con OpenSSL

- Para obtener un resumen **SHA1** de un archivo llamado fichero.txt:

```
# openssl dgst -sha1 fichero.txt
```

- Los resúmenes **MD5** y **SHA1** son idénticos a los creados con las órdenes **md5sum** y **sha1sum**, respectivamente, solo difieren en la salida.

Resúmenes con OpenSSL

```
# openssl dgst -md5 fichero.txt
# md5sum fichero.txt
```

```
# openssl dgst -sha1 fichero.txt
# sha1sum fichero.txt
```

Resúmenes con OpenSSL

- Podemos almacenar el resumen de un archivo usando la opción **out** de **OpenSSL**:

```
# openssl dgst -sha1 -out fichero.sha1
  fichero.txt
```

- Luego se puede hacer una verificación del resumen:

Resúmenes con OpenSSL

```
# openssl dgst -sha1 fichero.txt | cmp
  fichero.sha1
```

- Si no se despliega salida en pantalla, esto indica que el archivo no ha sido modificado.

Resúmenes con OpenSSL

- Los resúmenes son usados comúnmente para proveer integridad a los archivos transmitidos, ellos además poseen propiedades de autenticación y no-repudio para algún tipo de datos a través de firmas digitales.
- Un resumen encriptado con la clave privada del autor del archivo es lo más equivalente a la firma digital del archivo; otros podrían verificar el resumen del archivo usando la clave pública del autor.

Resúmenes con OpenSSL

- **OpenSSL** permite la creación de un resumen firmado a través de:

```
# openssl dgst -sha1 -sign privada.key
  -out fichero.sign fichero.txt
```

- Si lo que se desea es verificar el archivo fichero.txt junto con su firma **fichero.sign**, se necesita la clave pública **publica.key** y el archivo en la misma forma de cuando el resumen fue obtenido.

Resúmenes con OpenSSL

```
# openssl  dgst  -sha1  -verify
  publica.key       -signature
  fichero.sign fichero.txt
```

- La salida puede ser solamente **Verified OK** o **Verification Failure**.

Cifrado de contraseñas con OpenSSL

- Cifrar una contraseña con MD5 (opción **-1**):

```
# echo contraseña | openssl passwd -stdin -1
```

- Generar una contraseña con **crypt**:

```
# echo contraseña | openssl  passwd  -stdin   -crypt
```

Cifrado de contraseñas con OpenSSL

- Un **salt** es un texto adicional que se agrega a una contraseña.
- Se concatenan el **salt** y la contraseña para luego aplicar el **algoritmo de hash**. De esta forma se le agrega variación a las contraseñas pobremente definidas (contraseñas débiles).
- El texto de **salt** se debe crear con caracteres generados al azar.

Cifrado de contraseñas con OpenSSL

- Generar una contraseña con **crypt** y con **salt** KK:

```
# echo contraseña | openssl  passwd  -
   stdin   -crypt -salt KK
```

Resumen

- La