Seguridad en Redes de Comunicación

Práctica I. Introducción a los Mecanismos de Seguridad

Jorge Lanza Calderón

Luis Sánchez González

Departamento de Ingeniería de Comunicaciones

Este tema se publica bajo Licencia:
Creative Commons BY-NC-SA 4.0

Práctica 1

Introducción a los mecanismos de seguridad

1. Objetivo de la práctica

El objetivo de esta práctica es familiarizarse con las funciones criptográficas más comunes empleadas para proporcionar autenticación, confidencialidad, integridad y no repudio en las redes de comunicaciones.

- Uso de las herramientas proporcionadas por OpenSSL para facilitar la realización de operaciones criptográficas.
- Operativa del cifrado de clave simétrica y asimétrica empleando diferentes algoritmos.
- Análisis de metodologías de firma y verificación .

2. Introducción

El intercambio de información entre personas o entre cualquier entidad es una de las tareas que se realiza de forma cotidiana. En muchas de las ocasiones la información transferida es intrascendente y no requiere la aplicación de medidas de seguridad. Sin embargo, en los casos en los que la información es considerada sensible, es necesario tomar precauciones para evitar el acceso y uso a aquellos para los que no está destinada.

Existen diversas técnicas criptográficas que permiten hacer la información accesible únicamente a las partes deseadas, manteniéndola oculta para el resto. Estos mecanismos se diseñaron sobre la base de algoritmos de criptografía de clave simétrica y de clave asimétrica, estos últimos también conocidos como algoritmos de clave pública.

La implementación de éstos de forma óptima y eficiente no resulta sencilla. En la actualidad existen multitud de herramientas para cifrar datos o comunicaciones aplicando criptografía tanto simétrica como asimétrica.

El proyecto OpenSSL es una iniciativa de código libre basada en SSLeay, desarrollado por Eric Young y Tim Hudson, que ha dado lugar a un completo paquete de herramientas de administración y librerías con funciones criptográficas. Se trata de un software multiplataforma en constante actualización que permite la realización de, entre otras, las siguientes operaciones:

- Calcular funciones resumen de mensajes (MD5, SHA, etc.).
- Cifrar y descifrar con distintos algoritmos de cifrado.
- Generar claves RSA, DH y DSA.
- Gestionar el ciclo de vida de certificados X.509.
- Evaluar la implementación de clientes y servidores seguros.

3. Entorno de desarrollo

La práctica se desarrolla en una máquina ejecutando el sistema operativo *Ubuntu*. Se utilizará el usuario `alumnos`, con contraseña `telematica`. Debido a la configuración del entorno en el que se realiza la práctica, los alumnos deben guardar los archivos en los que hayan trabajado para no perder los cambios, y poder empezar con ellos durante la sesión siguiente.

A lo largo de esta práctica se emplearán las herramientas implementadas por OpenSSL a través de la línea de comandos usando el programa `openssl`. La mejor manera de comprender sus posibilidades y operativa es analizar su manual.

```
$ man openssl
```

Cada una de las funcionalidades de `openssl` tiene su propio manual accesible de manera similar a la ayuda global.

4. Desarrollo

Tras conocer las posiblidades de `openssl`, se pide generar un fichero de texto que contendrá la información que se desea asegurar y con el que se trabajará durante el resto de la práctica. Emplear cualquiera de las herramientas disponibles (`echo`, `cat`, `vi`, `gedit`, etc.)

4.1. Criptografía simétrica

Las operaciones de cifrado de criptografía simétrica se realizan empleando el comando `enc`, mediante el cual se puede cifrar y descifrar con diversos algoritmos de bloque o de flujo.

Generar una secuencia aleatoria que servirá de base para la definición de una clave suficientemente robusta para emplearla en las operaciones posteriores. Se podrá emplear igualmente una clave generada de forma manual.

```
$ dd if=/dev/urandom of=randombytes.bin bs=100 count=1
$ xxd randombytes.bin

$ cat /dev/urandom | head -1 > randombytes.bin
$ xxd randombytes.bin | head -1
```

El algoritmo AES permite trabajar con claves de 128 bits, 192 bits o 256 bits. Para la realización de esta práctica se empleará una longitud de clave de 128 bits.

Realizar el cifrado del fichero de texto en claro generado anteriormente empleando el algoritmo AES-128 en modo ECB.

```
$ openssl enc <mode> -in <plaintextfile> -out <ciphertextfile>
   -nopad -nosalt -iv <iv> -K <key>
$ xxd -b <ciphertextfile>
```

Analizar el contenido del fichero con el texto cifrado y comprobar si los parámetros del resultado obtenido coinciden con lo esperado.

Cuestión I

1) ¿Es relevante que la paridad de cada byte de la clave sea impar? Si en lugar de emplear el algoritmo AES se empleara el algoritmo DES, ¿sería relevante la paridad de la clave? Validar la respuesta de forma experimental.

2) ¿Cúal es la longitud del fichero de entrada? ¿Y del de salida?

3) A partir del fichero original, generar al menos 3 ficheros con una longitud igual al tamaño del bloque empleado en el algoritmo AES-128 y realizar el cifrado de éstos. ¿Cúal es el resultado obtenido? Compárelo con el resultado obtenido al cifrar el contenido de los 3 ficheros de forma conjunta.

Realizar la decodificación del fichero generado.

```
$ openssl enc <mode> -d -in <ciphertextfile> -out
  <deciphertextfile> -nopad -nosalt -iv <iv> -K <key>
$ xxd -b <deciphertextfile>
$ cat <deciphertextfile>
$ diff <deciphertextfile> <plaintextfile>
```

Analizar el contenido del fichero obtenido y comprobar si el resultado obtenido coincide con lo esperado.

Cuestión II

1) Modifique el contenido de uno de los bytes del primer bloque del fichero cifrado. ¿Es posible obtener el contenido del fichero original?

2) Modifique el contenido de uno de los bytes de cualquier otro bloque del fichero cifrado. ¿Es posible obtener el contenido del fichero original? Indique las diferencias que observa con respecto al caso anterior.

El tratamiento de ficheros binarios requiere del empleo de herramientas específicas para su visualización, lo que dificulta la transmisión de su contenido según el servicio que se emplee, como por ejemplo incrustado en el cuerpo de un correo electrónico. La codificación Base64 permite la representación de datos binarios usando únicamente los caracteres imprimibles de ASCII.

```
$ openssl enc -base64 -in file.txt -out file.b64
```

Cuestión III

1) Realizar el envío de la clave y el texto cifrado a través de correo electrónico a uno de los compañeros de forma que pueda decodificar el contenido del mensaje original.

2) Realizar el cifrado de al menos 3 de los bloques del fichero de texto en claro usado empleando el algoritmo AES-128 en modo CBC con un vector de inicialización todo ceros. Basarse en el algoritmo AES-128 en modo ECB y emplear la función `xor` proporcionada en el apéndice A y el comando `xxd` en su modo `revert`.

```
$ source bash_xor.txt
$ xor 0102030405060708 0102030405060708

$ echo -n 0102030405060708 | xxd -p -r > out.bin
$ xxd out.bin
0000000: 0102 0304 0506 0708 ........
```

   Comprobar el proceso anterior empleando la implementación del algoritmo AES-128 CBC existente en OpenSSL.

3) Modifique el contenido de uno de los bytes del primer bloque del fichero original y, empleando la implementación del algoritmo AES-128 CBC existente en OpenSSL, cifre dicho fichero. Repita el proceso modificando otro cualquiera de los bloques del fichero original. Indique las diferencias que observa con respecto al caso en el que se emplea AES-128 ECB (Cuestión I).

4.2. Funciones resumen

Las funciones de resumen criptográficas se realizan empleando el comando `dgst`, que también permite la realización de firmas electrónicas y su verificación. OpenSSL habilita la utilización de los diferentes algoritmos de forma directa como comandos.

Determinar el tamaño del resultado de aplicar las funciones de resumen MD5, SHA1, SHA256, etc. sobre el fichero de texto original.

```
$ openssl dgst <algorithm> <file>
```

Modificar el contenido del fichero y comparar el resultado con el obtenido anteriormente. Se puede usar el comando `diff`.

Cuestión IV

1) Para las diversas funciones de resumen, indicar cuál es el tamaño de el resumen obtenido para ficheros de longitud variable entre 10 bytes y 100 Kb.

4.3. Criptografía asimétrica

Las funcionalidades de criptografía asimétrica se encuentran repartidas entre diferentes comandos dentro de OpenSSL. En esta parte se estudiará la generación y gestión de la pareja de claves (`genpkey` o `pkey`) y las operaciones básicas de cifrado y generación de firma (`pkeyutl` o `dgst`).

Generar una pareja de claves pública y privada. Comenzar generando la clave privada para el algoritmo RSA.

```
$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048
   -pkeyopt rsa_keygen_pubexp:3 -out privkey-A.pem -des3
```

La clave privada debe estar protegida para que únicamente pueda ser accesible por su dueño. Es por esto que el comando anterior requiere la introducción de una clave de usuario. Sin embargo, para facilitar la realización de la práctica se recomienda eliminar dicho cifrado.

```
$ openssl pkey -in privkey-A.pem -out privkey-A-nociphered.pem
```

Observar el contenido del fichero de la clave privada en los dos casos anteriores, sabiendo que por defecto se genera en formato PEM.

```
$ cat privkey-A.pem
$ cat privkey-A-nocipher.pem
$ openssl pkey -in privkey-A-nociphered.pem -inform PEM -text
```

Realizar la transformación de formato PEM a formato DER. Para ello se empleará la funcionalidad de conversión de binario a Base64 y viceversa. Comprobar mediante el comando `pkey` que los valores obtenidos coinciden con lo observado anteriormente.

```
$ openssl base64 -d -in privkey-A.b64 -out privkey-A.bin
$ openssl asn1parse -in privkey-A.pem
$ openssl asn1parse -inform DER -in privkey-A.bin
$ openssl pkey -in privkey-A.bin -inform DER -text
```

Mediante OpenSSL se puede realizar la transformación de forma directa.

```
$ openssl pkey -in privkey-A.pem -out privkey-A.der -outform
  DER
$ openssl pkey -in privkey-A.der -inform DER -text
```

Generar ahora la clave pública a partir de la clave privada.

```
$ openssl pkey -in privkey-A.pem -out pubkey-A.pem -pubout
$ openssl pkey -in pubkey-A.pem -pubin -text
```

A continuación firmar digitalmente el fichero de texto generado al inicio de la práctica. Generar el hash del fichero y realizar la operación de cifrado con la clave correspondiente.

```
$ openssl dgst <alg> -binary <file> > <hashfile.bin>
$ openssl pkeyutl -sign -in <hashfile.bin> -inkey <key-A.pem>
  -out signature.bin -pkeyopt digest:<alg>
```

OpenSSL incluye otro modelo de operación que permite realizar el procedimiento en un solo paso.

```
$ openssl dgst <alg> -sign <key-A.pem> -out signature.bin
  <file>
```

Realizar el proceso de verificación de la firma, tanto sin modificar como modificando la información original.

```
$ openssl dgst -sha1 -verify <key-A.pem> -signature
  <signature.bin> <receivedfile>
```

Realizar el cifrado del fichero inicial.

```
$ openssl pkeyutl -encrypt -in <file> -pubin -inkey
  <key-A.pem> -out <ciphertext.bin>
```

Descifrar la secuencia codificada obtenida y comprobar que se obtiene el valor original.

```
$ openssl pkeyutl -decrypt -in <ciphertext.bin> -inkey
  <key-A.pem> -out <deciphertext.bin>
```

Cuestión V

1) ¿Puede tener el fichero a cifrar o descifrar cualquier longitud? ¿Cúal es la longitud máxima del fichero de entrada?

2) Si existiera alguna limitación, ¿cúal sería un posible procedimiento para poder transferir la información de manera segura entre dos entidades?

4.4. Intercambio seguro de información

A lo largo de esta práctica se ha trabajado con las herramientas proporcionadas por OpenSSL para cifrar y descifrar, así como para firmar y verificar información. Para ello se han empleado diferentes técnicas de cifrado de clave simétrica y asimétrica y funciones de resumen.

La aplicación de estos procedimientos permite proteger la transferencia de información sensible entre diferentes entidades.

Cuestión VI

1) Empleando los conocimientos adquiridos en esta práctica se pide definir, implementar y validar un procedimiento mediante el cual se realice el envío seguro de información entre dos nodos, garantizando la confidencialidad, la integridad y la autenticación.

5. Evaluación

Será necesario entregar una pequeña memoria descriptiva de los tareas realizadas durante la práctica en la que se muestre la resolución de las cuestiones planteadas.

Haciendo uso del procedimiento planteado en la cuestión VI, remitir los resultados al profesor responsable por correo electrónico. Para evitar posibles errores en el proceso y que sea imposible evaluar el resto de las cuestiones, se sugiere se remitan primeramente los resultados de forma no segura.

Se podrá realizar un examen tipo test tras la práctica o bien incluir alguna pregunta en el examen final con el objetivo que evaluar la comprensión de los conceptos trabajados.

> *Nota: Los asuntos de los correos electrónicos relacionados con esta práctica deberán contener la cadena* `[SRC15_PR1]` *seguida de la descripción breve del mensaje.*

A Función XOR para *bash*

```
# BASH function to get the result
# of a ^ b when a, b are in the
# following hexadecimal string
# form: AF396463D8705 ...

# Obtained from here:
# http://www.codeproject.com/Tips/470308/XOR-Hex-Strings-in-Linux-Shell-Script
# Author is Sanjay1982 (see http://www.codeproject.com/Members/Sanjay1982)

# Usage:
# $ xor AB20FF40 DD14FABC

function xor()
{
  local res=(`echo "$1" | sed "s/../0x& /g"`)
  shift 1
  while [[ "$1" ]]; do
      local one=(`echo "$1" | sed "s/../0x& /g"`)
      local count1=${#res[@]}
      if [ $count1 -lt ${#one[@]} ]
      then
            count1=${#one[@]}
      fi
      for (( i = 0; i < $count1; i++ ))
      do
            res[$i]=$((${one[$i]:-0} ^ ${res[$i]:-0}))
      done
      shift 1
  done
  printf " %02x" "${res[@]}"
}
```