Práctica 3: OpenSSL y Criptografía Pública

26 de noviembre de 2024

Índice

Resumen

El objetivo de esta práctica es la familiarización con el conjunto de herramientas OpenSSL, que es un paquete desarrollado por la comunidad OpenSource. Este paquete contiene una serie herramientas de administración y bibliotecas relacionadas con la criptografía. Por último se pretende que el alumno se familiarice con la criptografía pública, concretamente con el RSA. Este método de cifrado aprovecha el problema de la factorización de números enteros, y basa su funcionamiento en la potenciación de números mediante exponentes dados como producto de números primos de gran tamaño.

Introducción

La práctica puede ser realizada en lenguaje C de programación, o en Python. Se deberá elaborar una memoria sobre la práctica (en un fichero en formato pdf) que explique detalladamente la realización de todos los apartados con todos sus resultados correspondientes. En la memoria se podrá integrar el código necesario para entender la práctica correctamente. La elaboración de esta memoria es indispensable para la superación de la práctica. En el caso de la realización de la práctica en Python, se podrá presentar como memoria el PDF del "Jupyter Notebook — IPython"(explicando detalladamente la realización de todos los apartados con todos sus resultados correspondientes), a parte de los ficheros "*.ipynb"para poder ejecutar la práctica. En el caso de e la realización de la práctica en lenguaje C de programación, se presentarán los códigos de C utilizados más la memoria en PDF.

Problemas

1. OpenSSL

OpenSSL [1] es una librería para realizar operaciones básicas relacionadas con criptografía, gestión de certificados X.509 y sus formatos y protocolos SSL, TLS y S/MIME. Se puede utilizar a través de comandos (directamente o mediante shell) o mediante una librería de C. Vamos a ver a continuación unos cuantos ejemplos.

a. Cifrados simétricos

OpenSSL soporta una gran variedad de cifrados simétricos, a través de comandos o librería. Estudiar qué tipo de cifrados simétricos soporta. Con el comando `openssl help` se obtiene ayuda a un primer nivel de OpenSSL. Sobre esos comandos con `openssl -h` se obtiene un segundo nivel de ayuda, por ejemplo `openssl enc -h`. Para mas información utilizar el comando `man`. Con el comando `openssl ciphers -v`, se obtienen los posibles cifrados soportados por la herramienta. Explicar brevemente un poco cada uno de los cifrados soportados. Cifrar y descifrar con algunos de los algoritmos que existen comprobando su funcionamiento. Por ejemplo probar las claves débiles y semi-débiles del DES.

b. Cifrados asimétricos

Analiza que posibles cifrados asimétricos se pueden utilizar en openssl. Explica brevemente un poco cada uno de ellos. Cifrar y descifrar con algunos de los algoritmos comprobando su funcionamiento.

c. Generación de claves privadas y publicas

Estudiar en qué consiste un esquema de cifrado público basado en RSA y explicarlo brevemente. Generar una clave privada y pública con openSSL.

d. Diferencias entre velocidades de cifrados simétricos y asimétricos

Analiza cual es la diferencia en la velocidad en bytes procesados entre un cifrado simétrico y uno asimétrico. Para ello utiliza el comando `openssl speed`.

e. Certificados X.509

La criptografía de clave pública surgió fruto de la necesidad de resolver uno de los principales problemas en los esquemas de cifrado de información: la distribución de claves. Tras la formalización de Diffie-Hellman en base a *funciones de un sólo sentido*, dicho problema quedó resuelto. Ahora bien, tal formalización garantiza que un cierto usuario posee la clave privada correspondiente a una determinada clave pública, pero no aporta información explícita sobre el propietario de una pareja clave pública-clave privada. Por ello es necesario establecer un protocolo de autenticación de usuarios que haga factible vincular un usuario con un par clave pública-clave privada. La concreción de tal protocolo da origen a la llamada infraestructura de clave pública (en inglés, *Public Key Infrastructure* o PKI), en la cual se respalda la posesión y uso de una clave asimétrica por un usuario según un esquema horizontal o vertical de confianza. En el caso de PKIs con esquemas horizontales de confianza (como es el caso de PGP y de su versión *open-source* GPG), cada usuario acepta el uso de una clave asimétrica por parte de un tercer usuario en función del grado de confianza que el resto de usuarios manifiesta respecto al uso legítimo de dicha clave por parte del tercer usuario. Por otro lado, en los esquemas verticales de confianza (como el definido por el estándar X.509) existe una *tercera parte confiable o autoridad de certificación* que confirma (esto es, certifica) mediante un certificado (digital) la pertenencia de una cierta clave pública (y, consiguientemente, una clave privada) a un cierto usuario.

En esta práctica se va a trabajar en el contexto definido por la existencia de una autoridad de certificación que expide certificados digitales según el estándar X.509 y haciendo uso de openssl. Cada certificado X.509 contiene un número de versión y de serie, e información sobre los algoritmos utilizados para cifrar la información y para generar firmas digitales. La utilidad de estos certificados es posibilitar la seguridad de la capa de transporte (web, correo electrónico, redes privadas, etc). Explicar con un poco más de detalle qué utilidad pueden tener estos certificados. Generar un certificado X.509 auto-firmado (esto es, actuar como autoridad de certificación generando un certificado y firmándolo vosotros mismos) mediante openssl. Obtener información sobre el certificado generado a través de los comandos de openssl. Explicar claramente como funciona este certificado a nivel criptográfico mediante su protocolo de comprobación.

2. RSA (USO OBLIGATORIO DE GMP)

Para toda esta sección se recomienda consultar los capítulos 4, 8 y 14 del libro [2], así como el estándar del RSA [3].

a. Potenciación de grandes números

Programa una función que realice la potenciación de grandes números lo más rápidamente posible. La elección del algoritmo corre a cargo del grupo de prácticas, debiendo ser comentado en la memoria de la práctica, indicando las razones que han llevado a su elección.

Se realizará un programa que permita probar la funcionalidad implementada, contrastando los resultados obtenidos y las velocidades obtenidas con los de las funciones correspondientes de la biblioteca GMP(mpz_powm o mpz_powm_ui). El estudio realizalo con números de diferentes tamaños de bits: por ejemplo de tamaños de 100 bits hasta 4000 bits. Una posible interfaz para este programa podría ser la siguiente:

`potencia base exponente módulo`,

devolviendo por la salida estándar el resultado de ejecutar la operación "(base elevada a exponente) mod módulo".

b. Generación de números primos: Miller-Rabin

Desarrollar un programa de generación de números primos que compruebe la primalidad de un número mediante el algoritmo de Miller-Rabin, contrastando los resultados con las funciones correspondientes de la biblioteca GMP (mpz_probab_prime_p). Una posible interfaz para este programa podría ser la siguiente:

`primo -b bits -p sec [-o` $file_{out}$`]`,

donde:

-b número máximo de bits significativos que tendrá el número generado.

-p probabilidad de equivocación del algoritmo que especifica la seguridad que debería tener el primo generado en el test.

Con el parámetro de seguridad que se introduce en el programa, internamente deberás calcular el número de bases que tienes que generar de manera aleatoria para que tu numero primo generado tenga las seguridad de ser un número primo al nivel de confianza especificado.

Se devolverá por la salida especificada el número (aleatorio) candidato a primo, junto con su calificación, con el siguiente formato:

- Número candidato.

- Resultado de tu test.
- Resultado de GMP.
- Seguridad del primo: probabilidad de equivocación y número de veces que pasa el test, para generar un N primo de n bits.
- Tiempo que tardas en generarlo.

Basándote en este pequeño programa genera números primos de una tamaño arbitrario, por ejemplo 8096 bits, y comenta y estudia en la memoria la influencia del valor del parámetro -p en los resultados obtenidos. Este número primo grande que al final devolverá tu programa puede ser utilizado como una parte de una clave para el algoritmo RSA, cuando se necesite en la parte opcional del apartado de programación del RSA más adelante.

c. Factorización del módulo del RSA mediante el conocimiento de d (A. las Vegas)

Desarrollar un programa que dado que conoces el exponente de descifrado, sea capaz de calcular los dos números primos que componen el módulo del RSA, $n = p * q$. Probar que funciona dicho programa para módulos de RSA superiores a 4096 bits.

d. Números primos (OPCIONAL)

Hacer un análisis estadístico de los números aleatorios que pasan o no pasan el test de Miller-Rabin, y responder a las siguientes preguntas:

- Si un observador conoce el método aleatorio que usamos ¿podría determinar fácilmente el número escogido?
- ¿Se verifica la conjetura de Gauss de distribución de números primos (el número de primos menores o iguales a n se comporta como n/log(n))?
- Si el test ha sido positivo en n tests consecutivos ¿qué probabilidad hay de que continúe siéndolo en los m tests siguientes?

Las explicaciones deben razonarse de forma teórica apoyándose en las simulaciones experimentales por medio del algoritmo de Miller-Rabin.

e. Programación del RSA (OPCIONAL)

Implementa el método RSA, empleando la funcionalidad desarrollada en la realización de los apartados anteriores. El programa desarrollado presentará obligatoriamente la siguiente interfaz:

`RSA [Modos de funcionamiento]`

Donde los distintos modos son:

`-K` {`-k bits -t sec`} `[-o` $file_{out}$`]` : genera una clave aleatoria (es decir: el valor del módulo n, sus 2 factores primos p y q, y los valores de los exponentes d y e tales que $de = 1\ mod\ \phi(n)$ con los parámetros proporcionados, mostrándola en el fichero especificado.

`-C` {`-n mod -e` pub_{key} `|` `-f` $file_{clave}$} `[-i` $file_{in}$`]` `[-o` $file_{out}$`]` : cifra con los valores de n y e proporcionados (en base 10) o con los obtenidos del fichero de claves especificado.

`-D` {`-n mod -d` $priv_{key}$ `|` `-f` $file_{clave}$} `[-i` $file_{in}$`]` `[-o` $file_{out}$`]` : descifra con los valores de n y d proporcionados (en base 10) o con los obtenidos del fichero de claves especificado.

En la memoria de la práctica se comentará cómo se realiza el paso de fichero de texto plano (cifrado) a número RSA con el que cifrar (descifrar), a partir del valor proporcionado del parámetro -n. Comparar los resultado con OpenSSL.

Información complementaria

Comienzo grupo 1462: 26/11/2024, Entrega: 13/12/2024 (23:55 horas). La corrección será el 17/12/2024.

Comienzo grupo 1461: 28/11/2024, Entrega: 13/12/2024 (23:55 horas). La corrección será el 19/12/2024.

Bibliografía de referencia

[1] OpenSSL: `http://www.openssl.org/`

[2] Menezes, Alfred J. Handbook of applied cryptography Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. 1997 (`http://www.cacr.math.uwaterloo.ca/hac/`).

[3] PKCS #1 v2.1: RSA Cryptography Standard (`ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`).