Práctica 3
Criptografía asimétrica

En esta práctica utilizaremos las herramientas de OpenSSL para generar y utilizar claves y parámetros para criptosistemas asimétricos.

1. Manejo de claves pública y privada

Para generar un par pseudoaleatorio de claves pública y privada RSA se dispone del comando `openssl genrsa`[1], aunque en versiones recientes se recomienda usar el más genérico `openssl genpkey`[2] con el argumento `-algorithm RSA` (o con un archivo de parámetros que indique que se utilice RSA).

Para RSA, el número de bits de la clave generada y el exponente son por defecto 2048 y 65537. Estos valores pueden cambiarse con los argumentos adecuados.

Por defecto, los comandos imprimen por salida estándar la clave privada en texto plano en formato PEM[3], es decir, como una cadena en base64[4], precedida y sucedida por unas líneas que muestran que se trata de una clave privada sin cifrar. Añadiendo los argumentos pertinentes puede cifrarse la clave mediante AES o algún otro método de cifrado simétrico, junto con una contraseña (para generar la clave y el vector de inicialización a usar en el algoritmo simétrico) que puede pasarse como argumento, a través de archivo o a través de entrada estándar. La clave cifrada también se guarda en formato PEM, de forma que las líneas inicial y final indican que es una clave privada cifrada.

Ejercicio 1 (Generación de claves RSA)**.** Genera y guarda en un archivo con extensión `.pem` una clave RSA de 2048 bits en formato PEM cifrada con AES-128 (en el modo de operación que consideres) con una contraseña pseudoaleatoria (obtenida con `openssl rand`) o deliberadamente mala (como las vistas en la anterior sesión de prácticas). Presenta el contenido del archivo.

El comando `openssl pkey`[5] recibe como entrada una clave privada (posiblemente cifrada, en cuyo caso necesitaremos indicar la contraseña) o, con el argumento `-pubin`, pública. Puede ver los valores públicos (en caso de RSA, módulo y exponente público) y, si se le indicó una clave privada, privados (el resto de exponentes, el par de números primos...) con el argumento `-text` y comprobar la integridad de una clave privada con `-check`. Además, si no se utiliza `-noout`, al igual que `openssl genpkey` imprime la clave, opcionalmente a archivo y opcionalmente cifrada, por lo que puede utilizarse para cifrar con contraseña una clave privada o para cambiar o eliminar la contraseña de

[1] https://www.openssl.org/docs/manmaster/man1/openssl-genrsa.html
[2] https://www.openssl.org/docs/manmaster/man1/openssl-genpkey.html
[3] https://en.wikipedia.org/wiki/Privacy-Enhanced_Mail
[4] https://en.wikipedia.org/wiki/Base64
[5] https://www.openssl.org/docs/manmaster/man1/openssl-pkey.html

una clave ya cifrada, además de cambiar el formato si se utiliza el argumento `-outform`. También puede imprimir la clave pública a partir de una privada con el argumento `-pubout`. En general, si no queremos convertir la clave ni presentarla por pantalla, preferiremos utilizar el argumento `-noout`. El comando `openssl rsa`[6] hace lo mismo, pero solo funciona para claves RSA.

Ejercicio 2 (Conversión de formato)**.** Convierte (sin eliminarlo) el archivo obtenido en el ejercicio 1 en un archivo en formato DER y sin contraseña. Compara el tamaño de este archivo con el anterior.

Ejercicio 3 (Extracción de clave pública)**.** Obtén un archivo de clave pública en formato PEM a partir de la clave privada almacenada en el archivo obtenido en el ejercicio 1. Presenta el contenido del archivo.

A continuación, obtén un archivo de clave pública en formato PEM a partir de la clave privada almacenada en el archivo obtenido en el ejercicio 2 (requerirá utilizar el argumento `-inform DER` para reconocer correctamente el formato). Comprueba que el archivo obtenido es idéntico al anterior.

Ejercicio 4 (Visualización de valores)**.** Muestra los valores de la clave privada obtenida en el ejercicio 1 y los valores de la clave pública obtenida en uno cualquiera de los dos últimos ejercicios. Comprueba que estos últimos coinciden con los correspondientes valores de la clave privada.

2. Cifrado y firma con RSA

Para dar uso a las claves de RSA se dispone del comando `openssl rsautl`[7], desaconsejado en versiones recientes, y del más genérico `openssl pkeyutl`[8], que permite trabajar con otros modelos de criptografía asimétrica, infiriendo el modelo y el algoritmo a partir del archivo de clave que se le indique. Estos comandos permiten cifrar (`-encrypt`), descifrar (`-decrypt`), obtener firmas digitales (`-sign`) y comprobar firmas digitales (`-verify`, `-verifyrecover`; funcionan de forma distinta en `pkeyutl` y en `rsautl`).

Estos comandos asumen que reciben una clave privada a través del argumento `-inkey`, pero se puede indicar que es pública (o, en versiones recientes, usar la parte pública si es privada) mediante `-pubin`.

A la hora de cifrar y firmar, se asume que la entrada es suficientemente pequeña. En caso contrario, veremos un error. Lo habitual es cifrar una clave para un cifrado simétrico (o algo a partir de lo cual se pueda derivar una clave) y firmar el hash de un archivo en lugar del archivo completo, por lo que esta limitación de tamaño no suele ser un problema. Si se quiere firmar, en versiones recientes se puede utilizar `-rawin` para indicarle a OpenSSL que queremos que firme el hash del archivo, posiblemente junto con `-digest` para escoger el algoritmo de hash concreto; o podemos obtener manualmente el hash del archivo.

Ejercicio 5 (Firma y validación)**.** Obtén con `openssl dgst` el hash SHA-1 de este guion de prácticas y obtén una firma a partir de este hash con la clave privada obtenida en el ejercicio 1. Comprueba, utilizando `-verifyrecover` (en caso de utilizar el recomendado `pkeyutl`; si se utilizase `rsautl` sería con `-verify`), que se puede recuperar el hash a partir de la clave pública obtenida en el ejercicio 3. Comprueba también la firma digital

[6] https://www.openssl.org/docs/manmaster/man1/openssl-rsa.html
[7] https://www.openssl.org/docs/manmaster/man1/openssl-rsautl.html
[8] https://www.openssl.org/docs/manmaster/man1/openssl-pkeyutl.html

utilizando `-verify` (lo que requiere `pkeyutl` ya que en `rsautl` hace la anterior función) y la clave pública; observa que en este caso hay que indicar el archivo de firma a través de `-sigfile` en lugar de `-in`, que se usa para el hash.

Ejercicio 6 (Encapsulamiento de claves con RSA)**.** Genera un archivo aleatorio de 32 bytes (256 bits) llamado `secretocompartido.bin` utilizando `openssl rand`. Elige uno de los cifrados simétricos vistos en la práctica anterior y cifra con él el guion de esta práctica usando como contraseña el archivo aleatorio mediante el argumento `-pass file:secretocompartido.bin` (por lo que la clave y, si aplica, el vector de inicialización se derivarán del secreto compartido), y cifra el archivo `secretocompartido.bin` con la clave pública obtenida en el ejercicio 3.

Hecho esto, descifra con la clave privada obtenida en el ejercicio 1 el secreto cifrado y, usando el resultado, descifra el guion cifrado. Comprueba, utilizando funciones hash de OpenSSL, que el resultado final es idéntico al guion.

Ejercicio 7. Observa lo que sucede si se intenta cifrar o firmar (sin `-rawin`) un documento demasiado grande (como, por ejemplo, este guion o alguna memoria entregada anteriormente).

3. Intercambio de claves Diffie-Hellman

El comando `openssl pkeyutl` también permite la derivación de secretos compartidos a través de una clave privada y una pública no relacionada. Sin embargo, las claves RSA no son adecuadas para este propósito. Crearemos claves Diffie-Hellman para ello.

Para que ambas claves puedan combinarse en el intercambio de claves es necesario que procedan de unos parámetros comunes. En el caso de que las claves sean exponentes de un cierto generador en el conjunto de números enteros módulo un primo, es necesario que ambas claves consideren el mismo primo y el mismo generador.

Ejercicio 8 (Generación de parámetros)**.** Genera, utilizando `openssl dhparam`[9] o `openssl genpkey -genparam -algorithm DH`, un archivo de parámetros en formato PEM para unas claves Diffie-Hellman módulo un primo de 768 bits de longitud y tomando 2 como generador.[10]

Usando `openssl dhparam -check -text -noout -in` seguido del archivo obtenido, comprueba la integridad y visualiza los parámetros generados: la longitud (que debe ser 768 bits), el número primo pseudoaleatorio y el generador (que debe ser 2).

Ejercicio 9 (Intercambio de claves Diffie-Hellman)**.** Genera, usando el archivo de parámetros obtenido en el ejercicio anterior, *dos* pares de claves pública-privada Diffie-Hellman, cifrando las claves privadas con contraseñas pseudoaleatorias o deliberadamente malas, y después extrae con `openssl pkey` la parte pública de cada una. Visualiza con `openssl pkey -text` los valores de ambas claves privadas y ambas claves públicas, comprobando que el número primo es compartido por todas ellas, que cada clave pública aparece en su respectiva clave privada y que ambas claves privadas y ambas claves públicas son distintas entre sí.

Utilizando `openssl pkeyutl -derive`, genera un secreto compartido utilizando la primera clave privada (indicándola con `-inkey`) y la segunda clave pública (`-peerkey`). Comprueba que es el mismo secreto compartido que el obtenido si se utiliza la segunda

[9] https://www.openssl.org/docs/manmaster/man1/openssl-dhparam.html

[10] Nótese que todos estos ajustes (incluido el utilizar un grupo a partir de los números módulo un primo) son los que se toman por defecto excepto el número de bits, que es 2048.

clave privada y la primera clave pública, por lo que a través de este intercambio los dos propietarios de las claves privadas podrían comunicarse a través de un cifrado simétrico.

Ejercicio 10 (Replicando DH fuera de OpenSSL)**.** A partir del código desarrollado en la práctica 1:

- Convierte el número primo, las dos claves públicas y las dos claves privadas generadas en el último ejercicio de cadena hexadecimal (la que aparece con `-text` quitando los saltos de línea y los separadores `:` y posiblemente añadiendo `0x`) a números enteros de tamaño arbitrario del tipo desarrollado en la práctica 1.
- *Corrección de las claves.* Comprueba, utilizando la exponenciación modular rápida desarrollada en la práctica 1, que 2 (el generador) elevado a cada una de las claves privadas módulo el número primo devuelve un número idéntico a la correspondiente clave pública.
- *Generación de secreto compartido.* Comprueba, utilizando la exponenciación módulo el número primo, que la primera clave pública elevada a la segunda clave privada devuelve el mismo valor que la segunda clave pública elevada a la primera clave privada. Comprueba también que este valor, en hexadecimal, es idéntico al secreto compartido obtenido en el ejercicio anterior cuando este se visualiza en hexadecimal.

4. Curvas elípticas

A partir de curvas elípticas también es posible desarrollar protocolos de firma y verificación y de intercambio de claves. En el primer caso, las claves pueden generarse directamente a través de `openssl genpkey -algorithm EC`, de forma análoga a como se hizo con RSA, mientras que para hacer intercambio de claves habría que generar un archivo de parámetros y extraer múltiples claves del mismo, como se vio en el ejercicio 8, con `openssl genpkey -genparam -algorithm DH` o con el específico `openssl ecparam`[11], con el que también se puede generar una clave directamente con `-genkey -noout`, de forma que `-noout` impide que se escriban los parámetros y `-genkey` genera la clave privada.

Para indicar la curva elíptica a usar se utiliza `-pkeyopt ec_paramgen_curve:[nombre]` en `openssl genpkey` o `-name [nombre]` en `openssl ecparam`. Puede verse una lista de los nombres de curvas elípticas disponibles en OpenSSL junto con una breve descripción de cada una utilizando `openssl ecparam -list_curves`.

El comando `openssl ec`[12] permite procesar los archivos de claves de curvas elípticas. Dado que OpenSSL por defecto guarda el nombre que identifica a la curva elíptica a utilizar en las claves en lugar de los parámetros que la caracterizan, se puede utilizar `-param_enc explicit` al ver (con `-text -noout`) o al guardar a un archivo de claves o parámetros (sin `-text -noout`) los parámetros de una clave privada con `openssl ec` o los de una curva con `openssl ecparam`, que también permite visualizar los parámetros de una curva elíptica sin generar claves ni archivos de parámetros si se le pasan los argumentos `-name [nombre] -text -noout`. En versiones recientes de OpenSSL también puede utilizarse `-ec_param_enc explicit` para el mismo propósito en los genéricos `openssl pkey` y `openssl genpkey`.

[11] https://www.openssl.org/docs/manmaster/man1/openssl-ecparam.html
[12] https://www.openssl.org/docs/manmaster/man1/openssl-ec.html

Ejercicio 11 (Generación de claves a partir de curvas elípticas)**.** Genera una clave privada cifrada con una contraseña pseudoaleatoria o deliberadamente mala a partir de una curva elíptica sobre un cuerpo primo de al menos 200 bits y extrae su correspondiente clave pública. Muestra los valores de ambas claves, incluyendo los parámetros de la curva elíptica.

Después, utilizando `openssl ecparam`, obtén directamente los parámetros explícitos de la curva elíptica utilizada sin generar una clave en el proceso y comprueba que coinciden con los vistos en las claves anteriores.

Ejercicio 12 (Firma y verificación con curvas elípticas)**.** Repite el ejercicio 5 utilizando las claves generadas en el ejercicio anterior en lugar de las claves RSA. Es necesario utilizar el comando `openssl pkeyutl`; no hay análogo a `openssl rsautl` específico para curvas elípticas.