Sobre SMIME

Introducción

Nos ocupa ahora a utilidad `openssl-smime` o utilidad `smime - S/MIME` . `smime` es el acrónimo de *Secure Multipurpose Internet Mail Exchange* y es capaz de cifrar, descifrar, firmar y verificar mensajes `S/MIME` .

Tutorial de Urgencia

Hay seis opciones de la operación que determinan el tipo de operación que será llevada a cabo. El significado de las otras operaciones varía con el tipo de operación:

- `-encrypt` cifra un mensaje para los certificados de receptor dados. El fichero entrada en un mensaje para ser cifrado; el fichero de salida es el fichero cifrado en formato `MIME` . Sea observado que no es hecha ninguna comprobación de revocación para el certificado del receptor; así pues si esa clave ha sido comprometida, otros serán capaces de descifrar el texto.
- `-decrypt` descifra el mensaje usando el certificado suministrado y la clave privada; espera un mensaje cifrado en formato `MIME` como fichero de entrada. El mensaje descifrado es escrito en el fichero de salida.
- `-sing` firma un mensaje utilizando el certificado suministrado y la clave privada. El fichero de entrada es el mensaje a firmar. El mensaje firmado en formato `MIME` es escrito en el fichero de salida.
- `-verify` verifica mensajes firmados. Espera como entrada un mensaje de correo firmado y produce los datos de la firma. Están soportados texto llano y firma opaca.
- `-pk7out` toma un mensaje de entrada y proporciona escrito una estructura `PEM` cifrada `PKCS#7` .
- `-resign` vuelve a firmar un mensaje: toma un mensaje preexistente y uno o varios nuevos firmantes.
- `-in filename` donde `filename` es el mensaje de entrada que debe ser cifrado o firmado o el mensaje `MIME` que va a ser descifrado o verificado.
- `-inform SMIME|PEM|DER` esto especifica el formato de entrada para la estructura `PKCS#7` . El valor predeterminado es `SMIME` , que lee un mensaje de formato `S/MIME` . Los formatos `PEM` y `DER` cambian esto para esperar estructuras `PKCS#7` en formato `PEM` y `DER` . Esto actualmente sólo afecta al formato de entrada de la estructura `PKCS#7` , si no se está ingresando ninguna estructura `PKCS#7` (por ejemplo, con `-encrypt` o `-sign`) esta opción no tiene efecto.
- `-out filename` donde `filename` es el texto del mensaje que se ha descifrado o verificado o el mensaje de formato `MIME` de salida que se ha firmado o verificado.
- `-outform SMIME|PEM|DER` esto especifica el formato de salida para la estructura `PKCS#7` . El valor predeterminado es `SMIME` , que escribe un mensaje de formato `S/MIME` . Los formatos `PEM` y `DER` cambian esto para escribir estructuras `PKCS#7` en formato `PEM` y `DER` . Esto actualmente sólo afecta al formato de salida de la estructura `PKCS#7` , si no se está produciendo ninguna estructura `PKCS#7` (por ejemplo, con `-verify` o `-decrypt`) esta opción no tiene ningún efecto.
- `-stream` `-indef` `-noindef` las opciones `-stream` y `-indef` son equivalentes y habilitan streaming `I/O` para las operaciones de codificación. Esto permite el procesamiento de datos paso a paso sin la necesidad de mantener todo el contenido en la memoria, lo que potencialmente admite archivos de gran tamaño. Streaming se establece automáticamente para la firma `S/MIME` con datos separados si el formato de salida es `SMIME` , normalmente está desactivado de manera predeterminada para todas las demás operaciones.
- `-noindef` desactiva streaming `I/O` donde produciría una codificación de longitud indefinida. Esta opción normalmente no tiene efecto. En el futuro, streaming se habilitará por defecto en todas las operaciones relevantes y esta opción lo desactivará.
- `-content filename` especifica un archivo, de nombre `filename` , que contiene el contenido separado, esto sólo es útil con la orden `-verify` . Esto sólo se puede usar si la estructura `PKCS#7` está utilizando el formato de firma separado donde el contenido no está incluido. Esta opción anulará cualquier contenido si el formato de entrada es `S/MIME` y utiliza el tipo de contenido `MIME` firmado (de parte múltiple).

- `-text` esta opción agrega encabezados `MIME` de texto sin formato (texto/llano) al mensaje proporcionado, si está cifrando o firmando. Si está descifrando o verificando, elimina los encabezados de texto: si el mensaje descifrado o verificado no es de texto/llano tipo MIME, se produce un error.
- `-CAfile file` un fichero que contiene certificados fiables CA, sólo se usa con `-verify`.
- `CApath dir` un directorio que contiene certificados fiables de CA, sólo se usa con `-verify`. Este directorio debe ser un directorio de certificado estándar: es decir, un hash de cada nombre de sujeto (usando `x509 -hash`) debe estar vinculado a cada certificado.
- `-md digest` indicando un algoritmo de resumen para usar cuando se firma o refirma. Si no está presente, entonces será usado el algoritmo de resumen por defecto para la clave de firma (usualmente `SHA256`).
- `-[cipher]` el algoritmo de cifrado a usar. Por ejemplo, `DES` (56 bits) `-des`, `DES` triple (168 bits) `-des3`, puede ser usada una función `EVP_get_cipherbyname()` haciéndola preceder de un guión, por ejemplo `-aes_128_cbc`. Para conocer una lista de cifras soportadas por su versión de `OpenSSL` vea `enc`.
- `-nointern` al verificar un mensaje, normalmente los certificados (si los hay) incluidos en el mensaje son buscados para el certificado de firma. Con esta opción sólo se usan los certificados especificados en la opción `-certfile`. Sin embargo, los certificados suministrados todavía se pueden usar como CA no fiables.
- `-noverify` no verifica los certificados de firma de un mensaje firmado.
- `-nochain` no hace verificación en cadena de los certificados de los firmantes: esto es no usa los certificados en el mensaje firmado como CAs no fiables.
- `-nosigs` no intenta la verificación de las firmas en el mensaje.
- `-nocerts` cuando se firma un mensaje, el certificado del firmante se incluye normalmente; con esta opción, se excluye. Esto reducirá el tamaño del mensaje firmado, pero el verificador debe tener localmente una copia disponible del certificado de los firmantes (pasados mediante la opción `-certfile`, por ejemplo).
- `-noattr` cuando se firma un mensaje normalmente se incluye un conjunto de atributos como el tiempo de firma y los algoritmos simétricos admitidos; con esta opción no serán incluidos.
- `-binary` normalmente, el mensaje de entrada se convierte a formato "canónico" que utiliza efectivamente CR y LF como final de línea: como lo requiere la especificación S/MIME. Cuando esta opción está presente, no se produce ninguna traducción. Esto es útil cuando se manejan datos binarios que pueden no estar en formato MIME.
- `-nodetach` al firmar un mensaje usa firma opaca: este formato es más resistente a la traducción por retransmisión de correo, pero los agentes de correo que no son compatibles con `S/MIME` no pueden leerlo. Sin esta opción, se utiliza la firma de texto llano con el tipo `MIME` multipart/signed.
- `-certfile file` un certificado para firmar o refirmar un mensaje, esta opción se puede usar varias veces si se requiere más de un firmante. Si está siendo verificado un mensaje, los certificados de los firmantes se escribirán en este archivo si la verificación fue exitosa.
- `-signer file` un certificado de firma al firmar o refirmar un mensaje, esta opción se puede usar varias veces si se requiere más de un firmante. Si se verifica un mensaje, los certificados de los firmantes se escribirán en este archivo si la verificación fue exitosa.
- `-recip file` el certificado de los destinatarios al descifrar un mensaje. Este certificado debe coincidir con uno de los destinatarios del mensaje o se produce un error.
- `inkey file` la clave privada para usar al firmar o descifrar. Ésta debe coincidir con el certificado correspondiente. Si no se especifica esta opción, la clave privada debe incluirse en el archivo de certificado especificado con el archivo `-recip` o `-signer`. Al firmar, esta opción se puede usar varias veces para especificar claves sucesivas.
- `-passin arg` la fuente de la contraseña de la clave privada. Para más información sobre el formato de `arg` vea la sección de PASS PHRASE ARGUMENTS en openssl.
- `-rand file(s)` un archivo o archivos que contengan datos aleatorios utilizados para inicializar el generador de números aleatorios o un socket EGD (ver RAND_egd). Se pueden especificar varios archivos separados por un carácter dependiente del sistema operativo. El separador es `;` para MS-Windows, `,` para OpenVMS y `:` para todos los demás.
- `cert.pem...` uno o más certificados de receptores del mensaje: usado al cifrar un mensaje.

- `-to`, `-from`, `-subject` los encabezados de correo relevantes. Estos se incluyen fuera de la parte firmada de un mensaje, por lo que pueden incluirse manualmente. Si firma, entonces muchos clientes de correo `S/MIME` comprueban que la dirección de correo electrónico del certificado de firmantes coincide con la especificada en From: address.
- `-purpose`, `-ignore_critical`, `-issuer_checks`, `-crl_check`, `-crl_check_all`, `-policy_check`, `-extended_crl`, `-x509_strict`, `-policy` `-check_ss_sig` `-no_alt_chains` establecen varias opciones de verificación de cadenas de certificado. Vea la página del manual para `verify`.

Observaciones

El mensaje MIME debe ser enviado sin líneas en blanco entre los encabezados y la salida. Algunos programas de correo agregarán automáticamente una línea en blanco. Comunicar (piping) directamente el mensaje con `sendmail` es una forma de lograr el formato correcto.

El mensaje proporcionado para firma o cifra debe incluir los encabezados `MIME` necesarios o muchos clientes `S/MIME` no lo mostrarán correctamente (si es que lo hacen). Puede usar la opción `-text` para agregar automáticamente encabezados de texto sin formato.

Un mensaje "firmado y cifrado" es aquel en el que un mensaje firmado se cifra. Esto puede producirse encriptando un mensaje ya firmado: vea la sección de ejemplos.

Esta versión del programa sólo permite un firmante por mensaje, pero verificará múltiples firmantes en los mensajes recibidos. Algunos clientes `S/MIME` se atragantan si un mensaje contiene múltiples firmantes. Es posible firmar mensajes "en paralelo" al firmar un mensaje ya firmado.

Las opciones `-crypt` y `-decrypt` reflejan el uso común en clientes `S/MIME`. Estrictamente hablando, estos procesos `PKCS#7` envuelven datos: los datos cifrados de `PKCS#7` se usan para otros fines.

La opción `-resign` usa un resumen de mensaje existente al agregar un nuevo firmante. Esto significa que los atributos deben estar presentes en al menos un firmante existente que usa el mismo resumen de mensaje o esta operación fallará.

Las opciones `-stream` y `-indef` activan el soporte para streaming experimental `I/O`. Como resultado, la codificación es `BER` utilizando codificación construida de longitud indefinida y ya no es `DER`. Streaming tiene soporte para la operación `-encrypt` y la operación `-sign` si el contenido no está separado.

Streaming siempre se usa para la operación `-sign` con datos separados, pero como el contenido ya no forma parte de la estructura `PKCS#7`, la codificación permanece `DER`.

Códigos de Salida

- `0`; la operación fue completamente exitosa.
- `1`; se produjo un error al analizar las opciones de la orden.
- `2`; uno de los archivos de entrada no se pudo leer.
- `3`; se produjo un error al crear el archivo `PKCS#7` o al leer el mensaje `MIME`.
- `4`; se produjo un error al descifrar o verificar el mensaje.
- `5`; el mensaje se verificó correctamente, pero se produjo un error al escribir los certificados de los firmantes.

Ejemplos

Crear un mensaje llano firmado:

```
openssl smime -sign -in message.txt -text -out message.txt.sgn \
    -signer /path/to/your/certificate.pem \
    -inkey  /path/to/your/secret-key.pem -text
```

Crear un mensaje firmado opaco:

```
openssl smime -sign -in message.txt -text -out message.txt.sgn \
    -signer /path/to/your/certificate.pem \
    -inkey  /path/to/your/secret-key.pem -nodetach
```

Crear un mensaje firmado, incluir varios certificados adicionales y leer la clave privada de otro fichero:

```
openssl smime -sign -in in.txt -text -out mail.msg \
    -signer mycert.pem -inkey mykey.pem -certfile mycerts.pem
```

Enviar un mensaje firmado bajo Unix directamente a `sendmail`, incluyendo cabeceras:

```
openssl smime -sign -in message.txt -text
    -signer /path/to/your/certificate.pem \
    -inkey  /path/to/your/secret-key.pem -text \
    -from steve@openssl.org -to someone@somewhere \
    -subject "Signed message" | sendmail someone@somewhere
```

Verificar un mensaje y extraer el certificado del firmante si se tiene éxito:

```
openssl smime -verify -text -in message.txt.sgn -out message.txt
```

Enviar un mensaje cifrado usando `AES` triple:

```
openssl smime -encrypt -in message.txt -from steve@openssl.org \
    -to someone@somewhere -subject "Encrypted message" \
    -aes-256-cbc user.pem -out mail.msg
```

Firmar un mensaje cifrado:

```
openssl smime -sign -in ml.txt -signer my.pem -text \
    | openssl smime -encrypt -out mail.msg \
    -from steve@openssl.org -to someone@somewhere \
    -subject "Signed and Encrypted message" -des3 user.pem
```

Nota: la orden de cifrado no incluye la opción `-text` porque el mensaje que está siendo cifrado ya tiene encabezados `MIME`.

Descifrar un mensaje:

```
openssl smime -decrypt -in mail.msg -recip mycert.pem -inkey key.pem
```

El resultado del formato de firma de ciertos navegadores es una estructura `PKCS#7` con el formato de firma separada. Puede usar este programa para verificar la firma por línea envolviendo la estructura codificada en base64 y rodeándola con:

```
 -----BEGIN PKCS7-----
-----END PKCS7-----
```

y usando la orden:

```
openssl smime -verify -inform PEM -in signature.pem -content content.txt
```

Alternativamente puede decodificar base 64 la firma y usar:

```
openssl smime -verify -inform DER -in signature.der -content content.txt
```

Crear un mensaje cifrado usando Camellia de 128 bit:

```
openssl smime -encrypt -in plain.txt -camellia128 -out mail.msg cert.pem
```

Añadir un firmante a un mensaje existente:

```
openssl smime -resign -in mail.msg -signer newsign.pem -out mail2.msg
```

Resumen Escueto sobre un caso básico de Firma y Cifrado

Firma y Cifrado

Edite el texto del cuerpo del mensaje y guárdelo en un archivo separado, digamos `message.txt`. Debe tener a mano un certificado `X.509` y la clave privada asociada; entonces ejecute la siguiente orden:

```
openssl smime -sign -in message.txt -out message.txt.sgn \
        -signer /path/to/your/certificate.pem \
        -inkey  /path/to/your/secret-key.pem -text
```

Si además quiere cifrar el correo electrónico, sométalo también a un cifrado con S/MIME:

```
openssl smime -encrypt -in message.txt.sgn -out message.txt.sgn.enc \
        /path/to/intended-operators/certificate.pem
```

Verificación y Descifrado

A la llegada el mensaje es descifrado, caso de que hubiera sido cifrado:

```
openssl smime -decrypt -in message.sgn.enc \
        -out message.sgn \
        -recip /path/to/operators/certificate.pem \
        -inkey /path/to/operators/private-key.pem \
        -text
```

Seguidamente la firma es validada y el mensaje leído:

```
openssl smime -verify -text -in message.sgn \
        -noverify -out message.txt
```

La anterior orden emitirá un informe de verificación y extraerá el texto del mensaje en el fichero `message.txt`. Finalmente, el receptor comprueba si el firmante es ciertamente el remitente

```
openssl smime -pk7out -in message.sgn | openssl pkcs7 -print_certs -noout
```

Esto dará como resultado el DN del sujeto firmante (por ejemplo, el nombre de la RA)

Ejemplo Real

Tanto Eve como Bob han obtenido las respectivas parejas de clave y certificado de la acreditada CA Alice Ltd. las cuales son:

- la clave privada de Eve: `eve.key.pem`
- el correspondiente certificado de Eve: `eve.cert.pem`
- la clave privada de Bob: `bob.key.pem`
- el correspondiente certificado de Bob: `bob.cert.pem`

Obviamente la clave privada de ambos es celosamente custodiada por su respectivo dueño. El certificado de ambos es público.

De Parte del Emisor Eve

Eve conoce el certificado de Bob y se dispone a mandarle un mensaje firmado y cifrado. Entonces procede así:

- Primero escribe el texto de su email en un fichero llamado `message.txt`. El contenido del mensaje es:

```
Esta noche comienzo la misión. Espero tu presencia en el punto P a
la hora H.
```

- Luego firma Eve el trascendental mensaje:

```
openssl smime -sign -in message.txt -out message.txt.sgn \
    -signer eve.cert.pem \
    -inkey eve.key.pem -text
```

- Ha sido creado así el mensaje firmado `message.txt.sgn` que ahora será cifrado mediante la siguiente orden:

```
openssl smime -encrypt -in message.txt.sgn -out message.txt.sgn.enc \
              bob.cert.pem
```

Seguidamente Eve envía el fichero `message.txt.sgn.enc` a Bob por cualquier medio, quien efectivamente lo recibe y dispone de `bob.key.pem` y de `bob.cert.pem`:

De Parte del Receptor Bob

- Con `message.txt.sgn.enc` en su mano, Bob procede en primer lugar a descifrarlo:

```
openssl smime -decrypt -in message.txt.sgn.enc \
    -out message.txt.sgn \
    -recip bob.cert.pem \
    -inkey bob.key.pem
```

- Bob procede a averiguar datos sobre la supuesta autoría de la firma del mensaje recibido y que acaba de descifrar:

```
openssl smime -pk7out -in message.txt.sgn | \
openssl pkcs7 -print_certs -noout
```

  obteniendo el siguiente diálogo:

```
subject=/CN=eve@example.com
issuer=/C=GB/ST=England/O=Alice Ltd/CN=emisor_common_name
```

que le indica que hay un certificado vinculado, el de Eve, y que dicho certificado fue emitido por la autoridad Alice.

- Ahora Bob verificará la firma y, de tener éxito esa verificación, leerá el mensaje. Para verificarlo, dado que no ha

intervenido la CA en este envío, usará la orden (nótese el curioso uso de `-noverify` junto al `-verify`, pero que no es contradictorio):

```
openssl smime -verify -text -in message.txt.sgn -noverify -out message.txt
```

y obtendrá el satisfactorio diálogo

```
Verification successful
```

Bob conoce ahora las instrucciones para esta noche.

Prescindir de la Autoridad de Certificación

Tanto emisor como receptor pueden generar su clave privada y su certificado. En efecto el emisor genera su clave privada con un módulo de 4096 bits mediante:

```
openssl genrsa -out userS.key 4096
```

y seguidamente genera él mismo su certificado:

```
openssl req -new -x509 -days 1825 \
          -subj "/C=ES/ST=Spain/L=Ab/O=Cd/CN=UserS" \
          -key userS.key -out userS.cert.pem
```

El receptor hace lo propio. Crea su clave privada:

```
openssl genrsa -out userR.key 4096
```

y emite su propio certificado:

```
opensslreq -new -x509 -days 1825 \
          -subj "/C=ES/ST=Spain/L=Ab/O=Cd/CN=UserR" \
          -key userR.key -out userR.cert.pem
```

Ahora el usuario emisor tiene acceso legítimo al certificado del receptor y lo usará para comunicarse con él.

Observación

Recuerde que si cuenta usted con un fichero `file.ext` al que el sistema tenga asignada una aplicación para abrirlo, podrá abrirlo desde la terminal normalmente (teniendo instalado lo necesario para abrir ventanas dede la terminal) con `open`:

```
open file.ext
```

En particular, si el fichero es un `.pdf` podrá ejecutar

```
open file.pdf
```

y normalmente aparecerá abierto con la aplicación que por defecto abre ficheros `.pdf` en su sistema. Alternativamente puede usar lo siguiente:

```
xdg-open file.pdf
```

También podríamos incluso haber hecho:

```
$ gio open file.pdf
```

si es que contamos en el sistema con la orden `open`.

Recuerde que la cheksum del fichero `file.ext` puede ser llevada a cabo así:

```
$ sha256sum file.ext
971166aa9d995922149023da8789a7b5c5ba737d984f05d84ef84e408a6921d1  file.ext
```

Ejercicio

De entre sus compañeros elija una pareja y haga ante el profesor, compartiendo pantalla y repartiendose las tareas, lo siguiente. Usando `smime` y dado un fichero `.pdf`, ingenie una forma de:

- firmarlo doblemente con dos certificados simulando que lo hacen dos personas,
- cifrarlo con una cifra de su gusto para otro usuario conocido,
- efectuar el envío de del fichero cifrado,
- tras la recepción, descifrarlo y verificar la firma y
- finalmente, abrirlo con un visor de ficheros `.pdf`.

Generará para todo esto sus propias claves o elegirá usar las que le provee el profesor.

Referencias

Información básica tomada del texto de:

- 1.0.2 manpages: smime
- Manual:Smime(1)
- `smime` en Manpage
- Explicación de `smime` por `uni-bayreuth`