Sobre `pkeyutl` y `rsautl`

Introducción

Nos ocupana ahora las utilidades `openssl-pkeyutl` o `pkeyutl` y `rsautl`. `pkeyutl` es el acrónimo de *public key algorithm utility* y la orden puede ser utilizada para llevar a cabo operaciones con llave pública usando algoritmos soportados. La utilidad `rsautl` puede firmar, verificar, cifrar y descifrar datos usando el algoritmo RSA.

Manual de Urgencia de `pkeyutl`: opciones de la orden

- `-in filename` esto especifica el nombre de archivo de entrada para leer los datos o la entrada estándar si esta opción no está especificada.

- `-out filename` especifica el nombre de archivo de salida para escribir en él o salida estándar por defecto.

- `-inkey file` el archivo de la clave de entrada; por defecto debe ser una clave privada.

- `-keyform PEM|DER` el formato de la clave: `PEM`, `DER` o `ENGINE`.

- `-passin arg` la fuente de la contraseña de la clave de entrada. Para obtener más información sobre el formato de `arg`, consulte la sección PASS PHRASE ARGUMENTS en openssl.

- `-peerkey file` el archivo par de clave, utilizado por las operaciones de derivación de clave (acuerdo).

- `-peerform PEM|DER` el formato del par de la clave: `PEM`, `DER` o `ENGINE`.

- `-engine id` la especificación de un motor (por su cadena única de identificación) hará que `pkeyutl` intente obtener una referencia funcional del motor especificado, lo inicializará si es necesario. El motor se establecerá como el predeterminado para todos los algoritmos disponibles.

- `-pubin` la entrada es de clave pública.

- `-certin` la entrada es un certificado que contiene una clave pública.

- `-rev` invierte el orden del buffer de entrada. Esto es útil para algunas bibliotecas (como CryptoAPI) que representan el búfer en formato little endian.

- `-sing` firma los datos de entrada y da salida al resultado firmado. Esto requiere una clave privada.

- `-verify` verifica los datos de entrada con el archivo de firma e indica si la verificación tuvo éxito o falló.

- `-verifyrecover` verifica los datos de entrada y da salida a los datos recuperados.

- `-encrypt` cifra los datos de entrada utilizando una llave pública.

- `-decrypt` descifra los datos de entrada utilizando una llave privada.

- `-derive` deriva un secreto compartido usando la clave par.

- `-kexdump` vuelca a hex los datos de salida.

- `-asn1parse` asn1parse los datos de salida; esto es útil cuando se combina con la opción `-verifyrecover` cuando se firma una estructura ASN1.

Observaciones

Las operaciones y opciones admitidas varían de acuerdo con el algoritmo clave y su implementación. Las operaciones y opciones de `OpenSSL` se indican más abajo.

A menos que se mencione lo contrario, todos los algoritmos admiten la opción `digest alg`, que especifica el resumen que se utiliza para firmar, verificar y verificar las operaciones de recuperación. El valor `alg` debe representar un nombre resumido como se usa en la función `EVP_get_digestbyname()`, por ejemplo, `sha1`. Este valor se usa sólo para verificar la sensatez de las longitudes de los datos pasados a `pkeyutl` y para crear las estructuras que componen la firma (por ejemplo, `DigestInfo` en firmas `RSASSA PKCS#1 v1.5`). En el caso de firmas RSA, ECDSA y DSA, esta utilidad no realizará hash en los datos de entrada, sino que utilizará los datos directamente como entrada del algoritmo de firma. Dependiendo del tipo de clave, tipo de firma y modo de relleno, las longitudes máximas de datos de entrada aceptables difieren. En general, con RSA, los datos firmados no pueden ser más largos que el módulo clave; en el caso de ECDSA y DSA, los datos no deben ser más largos que el tamaño del campo; de lo contrario, se truncarán en silencio al tamaño del campo.

En otras palabras, si el valor de `digest` es `sha1`, la entrada debe tener una codificación binaria de 20 bytes de la salida de función hash SHA-1.

Algoritmo RSA

El algoritmo RSA admite en general las operaciones `encrypt`, `decrypt`, `sign`, `verify` y `verifyrecover`. Sin embargo, algunos modos de relleno sólo admiten algunas de estas operaciones.

- `-rsa_padding_mode:mode`

  Esto establece el modo de relleno de RSA. Los valores aceptables para el modo son `pkcs1` para el relleno PKCS#1, `sslv23` para el relleno SSLv23, ninguno para ningún relleno, `oaep` para el modo OAEP, `x931` para el modo X9.31 y `pss` para PSS.

  En el relleno PKCS#1, si el resumen del mensaje no está configurado, los datos suministrados se firman o verifican directamente en lugar de utilizar una estructura DigestInfo. Si se establece un resumen, entonces se usa la estructura a DigestInfo y su longitud debe corresponder al tipo de resumen.

  Para el modo `oeap`, sólo se admite el cifrado y descifrado.

  Para `x931`, si se establece el tipo de resumen, se utiliza para formatear los datos del bloque; de lo contrario, el primer byte se usa para especificar el ID del resumen X9.31. `sign`, `verify` y `verifyrecover` pueden realizarse en este modo.

  Para el modo `pss`, solo se admiten `sign` y `verify` y se debe especificar el tipo de resumen.

- `rsa_pss_saltlen:len`

  Para el modo `pss`, sólo esta opción especifica la salt length. Se admiten dos valores especiales: -1 establece la salt length a la longitud de resumen. Cuando se firma -2, se establece la salt length en el valor máximo permitido. Cuando se verifica -2, la salt length se determina automáticamente en función de la estructura del bloque PSS.

Algoritmo DSA

El algoritmo DSA sólo admite operaciones de firma y verificación. Actualmente no hay opciones adicionales aparte de `digest`. Sólo se puede usar el resumen SHA1 y este resumen se asume por defecto.

Algoritmo DH

El algoritmo DH sólo admite la operación de derivación y no tiene opciones adicionales.

Algoritmo EC

El algoritmo EC admite las operaciones `sign`, `verify` y `derive`. Las operaciones `sign` y `verify` usan ECDSA y `derive` usa ECDH. Actualmente no hay opciones adicionales aparte de `digest`. Sólo se puede usar el resumen SHA1 y este resumen se asume por defecto.

Ejemplos

Firma de algunos datos usando clave privada:

```
openssl pkeyutl -sign -in file -inkey key.pem -out sig
```

Recuperación de los datos firmados (por ejemplo, si se usa una clave RSA):

```
openssl pkeyutl -verifyrecover -in sig -inkey key.pem
```

Verificación de la firma (por ejemplo, una clave DSA):

```
openssl pkeyutl -verify -in file -sigfile sig -inkey key.pem
```

Firma de datos usando un valor de resumen de datos (esto es actualmente válido sólo para RSA):

```
openssl pkeyutl -sign -in file -inkey key.pem -out sig -pkeyopt digest:sha256
```

Derivación de un valor secreto compartido:

```
openssl pkeyutl -derive -inkey key.pem -peerkey pubkey.pem -out secret
```

Manual de Urgencia de `rsautl`: opciones de la orden

La orden `rsautl` puede ser usada para firmar, verificar, cifrar y descifrar datos usando el algoritmo RSA. Las opciones son las siguientes:

- `-in filename` esto especifica el nombre del archivo de entrada para leer los datos o la entrada estándar si esta opción no está especificada.

- `-out filename` especifica el nombre del archivo de salida para escribir en él o salida estándar por defecto.

- `-inkey file` el fichero de clave de entrada; por defecto debería ser una clave privada RSA.

- `-pubin` el fichero de entrada es una clave pública RSA.
- `-certin` la entrada es un certificado conteniendo una clave pública RSA.
- `-sing` firma los datos de entrada y da como salida el resultado firmado. Requiere una clave privada RSA.
- `-verify` verifica el dato de entrada y da como salida los datos recuperados.
- `-encrypt` cifra los datos de entrada usando una clave pública RSA.
- `-decrypt` descifra los datos de entrada usando una clave privada RSA.
- `-pkcs`, `-oaep`, `-ssl`, `-raw` el relleno a usar: PKCS#1 v1.5 (predeterminado), PKCS#1 OAEP, relleno especial utilizado en SSL v2 anteriores compatibles handskakes, o sin relleno, respectivamente. Para las firmas, sólo se pueden usar `-pkcs` y `-raw`.
- `-hexdump` volcado hex de los datos de salida.
- `-asn1parse` asn1parse los datos de salida; esto es lo usual cuando se combina con la opción `-verify`.

Observación

Debido a que `rsautl` usa directamente el algoritmo RSA, sólo puede ser usado para firmar y verificar porciones pequeñas de datos.

Ejemplos

Usa detarminados datos usando una clave privada:

```
openssl rsautl -sign -in file -inkey key.pem -out sig
```

Recupera el dato firmado:

```
openssl rsautl -verify -in sig -inkey key.pem
```

Examina los datos firmados en crudo:

```
openssl rsautl -verify -in sig -inkey key.pem -raw -hexdump
```

Es posible analizar la firma de certificados usando esta utilidad junto con asn1parse. Considere el ejemplo autofirmado en `certs/pca-cert.pem`. Ejecutando `asn1parse` de la siguiente manera:

```
openssl asn1parse -in pca-cert.pem
```

El BIT STRING final contiene la firma real. Puede ser extraída con:

```
openssl asn1parse -in pca-cert.pem -out sig -noout -strparse 614
```

El certificado de clave pública puede ser extraído con:

```
openssl x509 -in test/testx509.pem -pubkey -noout >pubkey.pem
```

La firma puede ser analizada con:

```
openssl rsautl -in sig -verify -asn1parse -inkey pubkey.pem -pubin
```

Ejemplo Real

Pasos Completados por el Emisor

En primer lugar obtenemos una pareja de llaves pública/privada con la siguiente orden:

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

Si deseamos ver su contenido, podemos hacerlo con `cat`:

```
cat privkey-ID.pem
```

constatando que está expresada en base 64. Para ver los valores individuales de la clave privada, podemos ejecutar:

```
openssl pkey -in privkey-ID.pem -text
```

que muestra lo que pudimos ver con `cat` y otros parámetros. La generación de la llave pública compañera y volcado en un fichero se lleva a cabo mediante la orden:

```
openssl pkey -in privkey-ID.pem -out pubkey-ID.pem -pubout
```

Para ver ahora los valores individuales de la clave pública, podemos ejecutar:

```
openssl pkey -in pubkey-ID.pem -pubin -text
```

Crearemos un fichero con un mensaje para transmitir:

```
Equipo:dir user$ echo Le ruego que disponga de 15540.40 dólares de
mi cuenta, a la que tiene acceso, para adquirir para mí un Bitcoin
>> message-ID.txt
```

Obtenemos una huella hash del mensaje contenido en el fichero `message-ID.txt` y la guardamos en el fichero `sing-ID.bin`:

```
openssl dgst -sha256 -sign privkey-ID.pem -out sign-ID.bin message-ID.txt
```

El receptor cuenta previamente con su pareja de claves pública/privada, de ellas oculta la privada, contenida en `privkey-IDR.pem` y ha hecho pública la pública, contenida en el fichero `pubkey-IDR.pem`. Ahora es cifrado el mensaje con RSA para evitar su lectura indebida. Obsérvese que hacemos intervenir la clave pública del que será el legítimo receptor:

```
openssl pkeyutl -encrypt -in message-ID.txt -pubin -inkey \
    pubkey-IDR.pem -out ciphertext-ID.bin
```

Obsérvese que el cifrado con RSA debería ser usado con ficheros pequeños, con longitud menor al menos similar a la de una clave. Si necesita cifrar mensajes largos, debe usar cifras de llave simétrica.

Ahora enviamos al receptor la pareja de ficheros `ciphertext-ID.bin` y `sign-ID.bin`.

Pasos Completados por el Receptor

El primer paso que completa el receptor es descifrar el mensaje:

```
openssl pkeyutl -decrypt -in ciphertext-ID.bin \
        -inkey privkey-IDR.pem -out received-ID.txt
```

Pero no actuará sin tener constancia de que el mensaje que ha recibido proviene realmente de quien dice ser el emisario. Para ganar esa certeza, ejecutará la siguiente orden:

```
openssl dgst -sha256 -verify pubkey-ID.pem -signature \
```

```
        sign-ID.bin received-ID.txt
```

Al recibir un mensaje que diga `Verified OK` estará seguro de que el emisario es quien dice ser. Esta seguridad descansa, por supuesto, en la fortaleza que atribuye al algoritmo `SHA-256`.

Ejercicio

Contando con la pública de mi pareja de llaves, haga lo siguiente:

- Cree una pareja de claves pública/privada RSA.
- Vea y comprenda los parámetros en la pareja de claves.
- Cree con `OpenSSL` un password de 32 caracteres escogidos entre: letras, mayúsculas y minúsculas, y dígitos del 0 al 9. No lo presente por pantalla, guárdelo en un fichero de nombre, digamos, `password-32.txt`.
- Con `sha256` y su clave privada obtenga una huella hash del fichero `password-32.txt`, que guardará en `sign-su-ID.bin`.
- Cifre el mensaje usando `pkeyutl` y la clave pública del receptor (el profesor) obteniendo el fichero `ciphertext-su-ID.bin`
- Envíe al profesor su firma del fichero que contiene el mensaje (`sign-su-ID.bin`) y el texto firmado ya cifrado (`ciphertext-su-ID.bin`). En este caso, como no está publicada, adjuntará a los dos ficheros su clave pública recién generada como parte de este ejercicio. Ruego que ponga los tres ficheros en una carpeta, la comprima y la suba a la plataforma.

Referencias

Información básica tomada del texto de:

- OpenSSL docs: pkeyutl
- OpenSSL docs: rsautl