NAME

openssl–pkeyutl, pkeyutl – public key algorithm utility

SYNOPSIS

openssl pkeyutl [**−help**] [**−in file**] [**−out file**] [**−sigfile file**] [**−inkey file**] [**−keyform PEM|DER|ENGINE**] [**−passin arg**] [**−peerkey file**] [**−peerform PEM|DER|ENGINE**] [**−pubin**] [**−certin**] [**−rev**] [**−sign**] [**−verify**] [**−verifyrecover**] [**−encrypt**] [**−decrypt**] [**−derive**] [**−kdf algorithm**] [**−kdflen length**] [**−pkeyopt opt:value**] [**−hexdump**] [**−asn1parse**] [**−rand file...**] [**−writerand file**] [**−engine id**] [**−engine_impl**]

DESCRIPTION

The **pkeyutl** command can be used to perform low level public key operations using any supported algorithm.

OPTIONS

−help
Print out a usage message.

−in filename
This specifies the input filename to read data from or standard input if this option is not specified.

−out filename
Specifies the output filename to write to or standard output by default.

−sigfile file
Signature file, required for **verify** operations only

−inkey file
The input key file, by default it should be a private key.

−keyform PEM|DER|ENGINE
The key format PEM, DER or ENGINE. Default is PEM.

−passin arg
The input key password source. For more information about the format of **arg** see the **PASS PHRASE ARGUMENTS** section in **openssl** (1).

−peerkey file
The peer key file, used by key derivation (agreement) operations.

−peerform PEM|DER|ENGINE
The peer key format PEM, DER or ENGINE. Default is PEM.

−pubin
The input file is a public key.

−certin
The input is a certificate containing a public key.

−rev
Reverse the order of the input buffer. This is useful for some libraries (such as CryptoAPI) which represent the buffer in little endian format.

−sign
Sign the input data (which must be a hash) and output the signed result. This requires a private key.

−verify
Verify the input data (which must be a hash) against the signature file and indicate if the verification succeeded or failed.

−verifyrecover
Verify the input data (which must be a hash) and output the recovered data.

−encrypt
Encrypt the input data using a public key.

−decrypt
Decrypt the input data using a private key.

−derive
Derive a shared secret using the peer key.

−kdf algorithm
Use key derivation function **algorithm**. The supported algorithms are at present **TLS1−PRF** and **HKDF**. Note: additional parameters and the KDF output length will normally have to be set for this to work. See **EVP_PKEY_CTX_set_hkdf_md**(3) and **EVP_PKEY_CTX_set_tls1_prf_md**(3) for the supported string parameters of each algorithm.

−kdflen length
Set the output length for KDF.

−pkeyopt opt:value
Public key options specified as opt:value. See NOTES below for more details.

−hexdump
hex dump the output data.

−asn1parse
Parse the ASN.1 output data, this is useful when combined with the **−verifyrecover** option when an ASN1 structure is signed.

−rand file...
A file or files containing random data used to seed the random number generator. Multiple files can be specified separated by an OS-dependent character. The separator is **;** for MS-Windows, **,** for OpenVMS, and **:** for all others.

[**−writerand file**]
Writes random data to the specified *file* upon exit. This can be used with a subsequent **−rand** flag.

−engine id
Specifying an engine (by its unique **id** string) will cause **pkeyutl** to attempt to obtain a functional reference to the specified engine, thus initialising it if needed. The engine will then be set as the default for all available algorithms.

−engine_impl
When used with the **−engine** option, it specifies to also use engine **id** for crypto operations.

NOTES

The operations and options supported vary according to the key algorithm and its implementation. The OpenSSL operations and options are indicated below.

Unless otherwise mentioned all algorithms support the **digest:alg** option which specifies the digest in use for sign, verify and verifyrecover operations. The value **alg** should represent a digest name as used in the **EVP_get_digestbyname()** function for example **sha1**. This value is not used to hash the input data. It is used (by some algorithms) for sanity-checking the lengths of data passed in to the **pkeyutl** and for creating the structures that make up the signature (e.g. **DigestInfo** in RSASSA PKCS#1 v1.5 signatures).

This utility does not hash the input data but rather it will use the data directly as input to the signature algorithm. Depending on the key type, signature type, and mode of padding, the maximum acceptable lengths of input data differ. The signed data can't be longer than the key modulus with RSA. In case of ECDSA and DSA the data shouldn't be longer than the field size, otherwise it will be silently truncated to the field size. In any event the input size must not be larger than the largest supported digest size.

In other words, if the value of digest is **sha1** the input should be the 20 bytes long binary encoding of the SHA−1 hash function output.

The Ed25519 and Ed448 signature algorithms are not supported by this utility. They accept non-hashed input, but this utility can only be used to sign hashed input.

RSA ALGORITHM

The RSA algorithm generally supports the encrypt, decrypt, sign, verify and verifyrecover operations. However, some padding modes support only a subset of these operations. The following additional **pkeyopt** values are supported:

rsa_padding_mode:mode

This sets the RSA padding mode. Acceptable values for **mode** are **pkcs1** for PKCS#1 padding, **sslv23** for SSLv23 padding, **none** for no padding, **oaep** for **OAEP** mode, **x931** for X9.31 mode and **pss** for PSS.

In PKCS#1 padding if the message digest is not set then the supplied data is signed or verified directly instead of using a **DigestInfo** structure. If a digest is set then the a **DigestInfo** structure is used and its the length must correspond to the digest type.

For **oaep** mode only encryption and decryption is supported.

For **x931** if the digest type is set it is used to format the block data otherwise the first byte is used to specify the X9.31 digest ID. Sign, verify and verifyrecover are can be performed in this mode.

For **pss** mode only sign and verify are supported and the digest type must be specified.

rsa_pss_saltlen:len

For **pss** mode only this option specifies the salt length. Three special values are supported: ''digest'' sets the salt length to the digest length, ''max'' sets the salt length to the maximum permissible value. When verifying ''auto'' causes the salt length to be automatically determined based on the **PSS** block structure.

rsa_mgf1_md:digest

For PSS and OAEP padding sets the MGF1 digest. If the MGF1 digest is not explicitly set in PSS mode then the signing digest is used.

RSA-PSS ALGORITHM

The RSA-PSS algorithm is a restricted version of the RSA algorithm which only supports the sign and verify operations with PSS padding. The following additional **pkeyopt** values are supported:

rsa_padding_mode:mode, **rsa_pss_saltlen:len**, **rsa_mgf1_md:digest**

These have the same meaning as the **RSA** algorithm with some additional restrictions. The padding mode can only be set to **pss** which is the default value.

If the key has parameter restrictions than the digest, MGF1 digest and salt length are set to the values specified in the parameters. The digest and MG cannot be changed and the salt length cannot be set to a value less than the minimum restriction.

DSA ALGORITHM

The DSA algorithm supports signing and verification operations only. Currently there are no additional **−pkeyopt** options other than **digest**. The SHA1 digest is assumed by default.

DH ALGORITHM

The DH algorithm only supports the derivation operation and no additional **−pkeyopt** options.

EC ALGORITHM

The EC algorithm supports sign, verify and derive operations. The sign and verify operations use ECDSA and derive uses ECDH. SHA1 is assumed by default for the **−pkeyopt digest** option.

X25519 and X448 ALGORITHMS

The X25519 and X448 algorithms support key derivation only. Currently there are no additional options.

EXAMPLES

Sign some data using a private key:

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

Recover the signed data (e.g. if an RSA key is used):

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

Verify the signature (e.g. a DSA key):

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

Sign data using a message digest value (this is currently only valid for RSA):

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

Derive a shared secret value:

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

Hexdump 48 bytes of TLS1 PRF using digest **SHA256** and shared secret and seed consisting of the single byte 0xFF:

```
openssl pkeyutl -kdf TLS1-PRF -kdflen 48 -pkeyopt md:SHA256 \
   -pkeyopt hexsecret:ff -pkeyopt hexseed:ff -hexdump
```

SEE ALSO

genpkey(1), **pkey**(1), **rsautl**(1) **dgst**(1), **rsa**(1), **genrsa**(1), **EVP_PKEY_CTX_set_hkdf_md**(3), **EVP_PKEY_CTX_set_tls1_prf_md**(3)

COPYRIGHT