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**] [**-engine id**] [**-engine_impl**]

DESCRIPTION

The **pkeyutl** command can be used to perform 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 and output the signed result. This requires a private key.

-verify
verify the input data against the signature file and indicate if the verification succeeded or failed.

-verifyrecover
verify the input data 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
asn1parse the output data, this is useful when combined with the **-verifyrecover** option when an ASN1 structure is signed.

-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 used only 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). In case of RSA, ECDSA and DSA signatures, this utility will not perform hashing on input data but rather use the data directly as input of signature algorithm. Depending on key type, signature type and mode of padding, the maximum acceptable lengths of input data differ. In general, with RSA the signed data can't be longer than the key modulus, in case of ECDSA and DSA the data shouldn't be longer than field size, otherwise it will be silently truncated to field size.

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

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. Two special values are supported: -1 sets the salt length to the digest length. When signing -2 sets the salt length to the maximum permissible value. When verifying -2 causes the salt length to be automatically determined based on the **PSS** block structure.

DSA ALGORITHM

The DSA algorithm supports signing and verification operations only. Currently there are no additional options other than **digest**. Only the SHA1 digest can be used and this digest is assumed by default.

DH ALGORITHM

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

EC ALGORITHM

The EC algorithm supports sign, verify and derive operations. The sign and verify operations use ECDSA and derive uses ECDH. Currently there are no additional options other than **digest**. Only the SHA1 digest can be used and this digest is assumed by default.

X25519 ALGORITHM

The X25519 algorithm supports 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