wolfCLU Documentation

2022-07-01

Contents

1 wolfCLU Manual

wolfSSL's Command Line Utility (version 0.0.7)
Nov, 24, 2021

1.1 Intro

wolfCLU was created to handle some common cryptographic operations to make it easier/quicker then writing an application from scratch. An example of some of the operations handled are certificate parsing and key generation.

1.2 Building wolfCLU

To build wolfCLU first build wolfSSL with the –enable-wolfclu flag. An example of this would be:

```
cd wolfssl
./configure --enable-wolfclu
make
sudo make install
```

Note that if parsing PKCS12 files with RC2 or if using CRL the flags –enable-rc2 and –enable-crl would also need to be used when building wolfSSL.

Then build wolfCLU linking agianst the wolfSSL library created.

```
cd wolfclu
./configure
make
sudo make install

or

cd wolfclu
./configure --with-wolfssl=/path/to/wolfssl/install
make
sudo make install
```

1.3 List Of Commands:

- bench
- crl
- dgst
- ecparam
- enc
- genkey
- hash
- md5
- pkcs12
- pkey
- rand
- req
- rsa
- s_client
- verify
- x509

1.3.1 BENCH Command

Command in progress for benchmarking algorithms. Current use to run all algorithms would be “wolfssl bench -all”.

1.3.2 CRL Command

Used to verify a CRL file given a CA. Or to convert a CRL from one format [DER | PEM] to the other. The command will also print out the CRL to stdout if -out is not specified and -noout is not used. Prints out “OK” on successful verification.

- [-CAfile]
- [-inform] pem or der in format
- [-in] the file to read from
- [-outform] pem or der out format
- [-out] output file to write to
- [-noout] do not print output if set

Example:

```
wolfssl crl -CAfile ./certs/ca-cert.pem -in ./certs/crl.der -inform DER -noout
```

1.3.3 DGST Command

Can verify the signature. The last argument is the data that was signed.

Hash algos supported:

- [-sha]
- [-sha224]
- [-sha256]
- [-sha384]
- [-sha512]

Parameters:

- [-signature] file containing the signature
- [-verify] key used to verify the signature

Example:

```
wolfssl dgst -signature test.sig -verify key.pem test
```

1.3.4 ECPARAM Command

Used for creating ECC keys.

Available arguments are:

- [-genkey] create new key
- [-out] output file
- [-name] curve name i.e. secp384r1

Example:

```
wolfssl ecparam -genkey -out new.key -name secp384r1
```

1.3.5 ENC Command

Used for encrypting an input and with (-d) can decrypt also.

Available encryption and decryption algorithms are:

- aes-cbc-128

- aes-cbc-192
- aes-cbc-256
- aes-ctr-128
- aes-ctr-192
- aes-ctr-256
- 3des-cbc-56
- 3des-cbc-112
- 3des-cbc-168

Available arguments are:

- [-in] input file to read from
- [-out] file to write to (default stdout)
- [-pwd] password input
- [-key] hex key input
- [-iv] hex iv input
- [-inkey] input file for key
- [-pbkdf2] use kdf version 2
- [-md] specify hash algo to use i.e md5, sha256
- [-d] decrypt the input file
- [-p] display debug information (key / iv ...)
- [-k] another option for password input
- [-base64] handle decoding a base64 input
- [-nosalt] do not use a salt input to kdf

Example:

```
wolfssl enc -aes-128-cbc -k Thi$i$myPa$$w0rd -in somefile.txt
```

1.3.6 GENKEY Command

Used to generate RSA, ECC, and ED25519 keys. If using "-output KEY" a private key is created having .priv appended to -out argument and a public key is created with .pub appended to the -out argument. If generating ED25519 keys compile wolfSSL with –enable-ed25519.

Available arguments are:

- [-out] file to write to
- [rsa | ecc | ed25519] key type to generate
- [-inkey] input file for key
- [-size] size of key to generate
- [-outform] output form, either DER or PEM
- [-exponent] RSA exponent size

Example:

```
wolfssl genkey rsa -size 2048 -out mykey -outform pem  -output KEY
```

1.3.7 HASH Command

Used to create a hash of input data.

Algorithms:

- md5
- sha
- sha256
- sha384
- sha512

- base64enc
- base64dec

Example:

```
wolfssl -hash sha -in <some file>
```

1.3.8 MD5 Command

Used to create a MD5 hash of input data. The last argument is the file to be hashed, if a file argument is not used then stdin is pulled for data to be hashed.

Example :

```
wolfssl md5 configure.ac
978425cba5277d73db2a76d72b523d48

echo "hi" | wolfssl md5
764efa883dda1e11db47671c4a3bbd9e
```

1.3.9 PKCS12 Command

Currently only PKCS12 parsing is supported and PKCS12 generation is not yet supported. By default the –enable-wolfclu option used when building wolfSSL has PKCS12 support also enabled but it does not enable RC2. If parsing PKCS12 bundles that have been encrypted using RC2 then –enable-rc2 should also be used when compiling wolfSSL.

- [-in] file input for pkcs12 bundle
- [-out] file to output results to (default stdout)
- [-nodes] no DES encryption
- [-nocerts] no certificate output
- [-nokeys] no key output
- [-passin] source to get password from
- [-passout] source to output password to

Example:

```
./wolfssl pkcs12 -nodes -passin pass:"wolfSSL test" -in ./certs/test-servercert.p12
```

1.3.10 PKEY Command

Used for dealing with generic key operations. Prints the key read in to stdout.

- [-in] file input for key
- [-inform] pem or der input format
- [-pubout] only print out the public key

Example:

```
./wolfssl pkey -in ./certs/server-key.pem -inform pem -pubout
```

1.3.11 RAND Command

Generates random bytes in raw or base64 form. By default it outputs the result to stdout but can be redirected with using the ‘-out’ argument. The last argument passed in is the number of random bytes to generate.

- [-base64] base64 encode the resulting random bytes
- [-out] ouput file to write results to

Example:

```
wolfssl rand -base64 10
```

1.3.12 REQ Command

Used for creating a certificate request or a self-signed certificate. Can handle some basic parsing of a .conf file for certificate setup. If no configuration file is used then stdin is prompted for certificate information.

Available arguments are:

- [-in] input file to read from
- [-out] file to write to (default stdout)
- [-key] public key to put into certificate request
- [-inform] der or pem format for '-in'
- [-outform] der or pem format for '-out'
- [-config] file to parse for certificate configuration
- [-days] number of days should be valid for
- [-x509] generate self signed certificate

Example:

```
wolfssl ecparam -genkey -out ecc.key -name secp384r1
wolfssl req -new -x509 -days 3650 -config selfsigned.conf -key ecc.key -out ecc.cert \
-outform der -sha256
```

1.3.13 RSA Command

Does RSA operations. Including reading in RSA keys, outputing RSA keys or modulus, and reading encrypted PEM files. Can handle both DER and PEM format for input and output. The following is a list of options

- [-in] file input for key to read
- [-inform] PEM or DER input format
- [-out] file to write result to (defaults to stdout)
- [-outform] PEM or DER output format
- [-passin] password for PEM encrypted files
- [-noout] do not print the key out when set
- [-modulus] print out the RSA modulus (n value)
- [-RSAPublicKey_in] expecting a public key input ### S_CLIENT Command Very basic TLS connection supported. Currently does not verify the peer, -CAfile option is not yet completed.

Arguments:

- [-connect] :

Example :

```
wolfssl s_client -connect 127.0.0.1:11111
```

1.3.14 VERIFY Command

Verifies an X509 certificate given a CA. The last argument passed into the command is the certificate file name to be verified. If the verification is successful then "OK" will be printed out to stdout. Otherwise an error value and reason will be printed out.

- [-CAfile] file name for CA to be used with verify
- [-crl_check] if CRL checking should be used

Example:

```
wolfssl verify -CAfile ./certs/ca-cert.pem ./certs/server-cert.pem
```

1.3.15 X509 Commnad

This command is used for parsing and printing out certificates.

Arguments:

- [-in] X509 file input
- [-inform] pem or der format for input
- [-out] file to output to
- [-outform] pem or der format for output
- [-pubkey] print out the public key only
- [-text] print out the certificate

Example:

```
wolfssl x509 -in ./certs/server-cert.pem -text
```