

wolfCLU Documentation



2026-01-19

Contents

1	wolfCLU Manual	3
1.1	Intro	3
1.2	Building wolfCLU	3
1.2.1	Building on *NIX	3
1.3	Building on Windows	3
1.4	List Of Commands:	4
1.4.1	BENCH Command	4
1.4.2	CA Command	4
1.4.3	CRL Command	5
1.4.4	DSAPARAM Command	5
1.4.5	DGST Command	5
1.4.6	DHPARAM Command	6
1.4.7	ECPARAM Command	6
1.4.8	ENC Command	7
1.4.9	GENKEY Command	7
1.4.10	HASH Command	8
1.4.11	MD5 Command	8
1.4.12	PKCS12 Command	8
1.4.13	PKCS7 Command	9
1.4.14	PKCS8 Command	9
1.4.15	PKEY Command	9
1.4.16	RAND Command	10
1.4.17	REQ Command	10
1.4.18	RSA Command	10
1.4.19	sha256, sha384, and sha512 commands	11
1.4.20	S_CLIENT Command	11
1.4.21	VERIFY Command	11
1.4.22	X509 Commnad	11
1.4.23	BASE64 Command	12

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 than writing an application from scratch. An example of some of the operations handled are certificate parsing and key generation.

1.2 Building wolfCLU

1.2.1 Building on *NIX

To build wolfCLU, start by building 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 against 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
```

Run `make check` to run unit tests.

1.3 Building on Windows

wolfCLU can also be built with the appropriate Visual Studio solution, `wolfclu.sln`. The solution provides both Debug and Release builds of Dynamic 32- or 64-bit libraries. The file `user_settings.h` should be used in the wolfSSL build to configure it.

The file `wolfclu\ide\winvs\user_settings.h` contains the settings used to configure wolfSSL with the appropriate settings. This file must be copied from the directory `wolfclu\ide\winvs` to `wolfssl\IDE\WIN`. You can then build wolfSSL with support for wolfCLU.

Before building wolfCLU, make sure you have the same architecture (Win32 or x64) selected as used in wolfSSL.

This project assumes that the wolfSSH and wolfSSL source directories are installed side-by-side and do not have the version number in their names:

```
Projects\  
  wolfclu\  
  wolfssl\
```

Building a wolfCLU release configuration will generate wolfssl.exe in the Release\Win32 or Release\x64 directory.

1.3.0.1 Running Unit Tests To run the shell-script unit tests, you will need either the sh or bash commands, both of which come with a Git installation on Windows (although you may have to add them to the PATH).

1. Copy the wolfssl.exe to the root directory of wolfclu.
2. Modify the run function (as well as run_fail, if present) of the desired unit test to run ./wolfssl.exe \$1 instead of ./wolfssl \$1.
3. In your terminal, run sh <desired_unit_test> from the root directory. For instance, to run the hash unit tests, run sh tests\hash\hash-test.sh.

1.4 List Of Commands:

- base64
- bench
- ca
- crl
- dsaparam
- dgst
- ecpparam
- enc
- genkey
- hash
- md5
- pkcs12
- pkcs7
- pkcs8
- pkey
- rand
- req
- rsa
- s_client
- verify
- x509
- dhparam
- sha256
- sha384
- sha512

1.4.1 BENCH Command

Command in progress for benchmarking algorithms. To benchmark all algorithms run "wolfssl bench -all".

1.4.2 CA Command

Used for signing Certificates. Can handle some basic config file parsing.

Available arguments are:

- [-in] input CSR file
- [-out] file to write to
- [-keyfile] file to read private key from
- [-cert] file to read CA from
- [-extensions] section in config file to parse extensions from
- [-md] type of hash to use i.e sha, sha256, ...
- [-inform] PEM/DER type of CSR input
- [-config] file to parse for configuration
- [-days] number of days should be valid for
- [-selfsign] sign with key associated with input cert

Example:

```
wolfssl ca -config ca.conf -in test.csr -out test.pem -md sha256 -selfsign -  
keyfile ./key
```

1.4.3 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 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
```

1.4.4 DSAPARAM Command

Used for creating DSA params and keys. Make sure wolfSSL is compiled with --enable-dsa.

Available arguments are:

- [-genkey] create new DSA key
- [-in] file to read params from to create a key
- [-out] file to output to (default stdout)
- [-noout] do not print out the params

Example:

```
wolfssl dsaparam -out dsa.params 1024
```

```
wolfssl dsaparam -in dsa.params -genkey
```

1.4.5 DGST Command

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

Hash algos supported:

- [-sha]
- [-sha224]

- [-sha256]
- [-sha384]
- [-sha512]

Sign

Parameters:

- [-sign] key used to create signature
- [-out] file to write signature to

Example:

```
wolfssl dgst -sign keyPrivate.pem -out test.sig testfile
```

Verify

Parameters:

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

Example:

```
wolfssl dgst -verify keyPublic.pem -signature test.sig testfile
```

1.4.6 DHPARAM Command

Used for creating Diffie Hellman params and keys.

Available arguments are:

- [-genkey] create new DH key
- [-in] file to read params from to create a key
- [-out] file to output to (default stdout)
- [-check] check if generated parameters are valid
- [-noout] do not print out the params

Example:

```
wolfssl dhparam -check -out dh.params 1024
```

1.4.7 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
```

1.4.8 ENC Command

Used for encrypting an input. Setting -d enables decryption.

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
```

1.4.9 GENKEY Command

Used to generate RSA, ECC, ED25519 and DSA 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 (defaults to DER)
- [-output] key to generate, either PUB, PRIV or KEYPAIR (defaults to KEYPAIR)
- [-exponent] RSA exponent size

Example:

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

1.4.10 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>
```

1.4.11 MD5 Command

Used to create an MD5 hash of input data. The last argument is the file to be hashed, if a file argument is not provided then stdin is used. Note that when using stdin the input must be provided upon calling wolfssl. A correct usage and incorrect usage of stdin are shown below:

Incorrect :

```
wolfssl md5
> hi
>
```

Correct :

```
echo "hi" | wolfssl md5
```

The reason the incorrect version is wrong is because no output will be received. This input method is used because it mimics the usage of OpenSSL's CLI, thus it encourages portability when switching from OpenSSL to WolfSSL.

Examples :

```
wolfssl md5 configure.ac
```

```
978425cba5277d73db2a76d72b523d48
```

```
echo "hi" | wolfssl md5
```

```
764efa883dda1e11db47671c4a3bbd9e
```

1.4.12 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 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
```

1.4.13 PKCS7 Command

Processes PKCS#7 data, allowing operations such as extracting certificates from PKCS#7 files. PKCS#7 is a standard for cryptographically signed and/or encrypted data.

- [-in] input file containing PKCS#7 data (required)
- [-out] output file to write results to (default stdout)
- [-inform] input format (PEM or DER, default PEM)
- [-outform] output format (PEM or DER, default PEM)
- [-print_certs] extract and output certificates from the PKCS#7 file

Example for extracting certificates from a PKCS#7 file:

```
wolfssl pkcs7 -in pkcs7.pem -print_certs
```

Example for converting PKCS#7 data from PEM to DER format:

```
wolfssl pkcs7 -in pkcs7.pem -outform DER -out pkcs7.der
```

1.4.14 PKCS8 Command

Processes PKCS#8 private key files. Allows conversion between different formats (PEM/DER) and decryption of private keys. Encrypting PKCS#8 keys is not yet supported.

- [-in] input file containing the private key (required)
- [-out] output file to write the processed key to (default stdout)
- [-inform] input format (PEM or DER, default PEM)
- [-outform] output format (PEM or DER, default PEM)
- [-passin] password source for encrypted input key
- [-traditional] output key in traditional (non-PKCS#8) format
- [-topk8] convert input to PKCS#8 format
- [-nocrypt] don't encrypt the output key (no password)

Example for converting an encrypted PEM key to DER format:

```
wolfssl pkcs8 -in server-keyEnc.pem -passin pass:mypassword -outform DER -out key.der
```

Example for converting a key to traditional format:

```
wolfssl pkcs8 -in server-key.pem -traditional -out traditional-key.pem
```

1.4.15 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 (defaults to pem)

- [-pubout] only print out the public key
- [-pubin] expect to public key as input

Example:

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

1.4.16 RAND Command

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

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

Example:

```
wolfssl rand -base64 10
```

1.4.17 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' (defaults to pem)
- [-outform] der or pem format for '-out' (defaults to pem)
- [-config] file to parse for certificate configuration
- [-days] number of days should be valid for
- [-x509] generate self signed certificate

Example:

```
wolfssl ecpkparam -genkey -out ecc.key -name secp384r1
```

```
wolfssl req -new -x509 -days 3650 -config selfsigned.conf -key ecc.key -out  
ecc.cert -outform der -sha256
```

1.4.18 RSA Command

Does RSA operations. Including reading in RSA keys, outputting 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 (defaults to PEM)
- [-out] file to write result to (defaults to stdout)
- [-outform] PEM or DER output format (defaults to PEM)
- [-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

1.4.19 sha256, sha384, and sha512 commands

Each command can be used to create a hash of its type. sha256 generates a sha256 hash and so on. The commands accept input in the form of stdin or a specified input file.

Example :

```
wolfssl -sha384 file.txt
```

```
echo "hi" | wolfssl -sha384
```

1.4.20 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
```

1.4.21 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
- [-untrusted] file name for intermediate certificate to be used in verification (only one -untrusted cert is currently supported)

Example:

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

1.4.22 X509 Command

This command is used for parsing and printing out certificates.

Arguments:

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

Example:

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

1.4.23 BASE64 Command

Encodes or decodes data using base64 encoding. By default, data is encoded to base64. Use the -d option to decode base64 data.

- [-in] input file to read from (default stdin)
- [-out] output file to write to (default stdout)
- [-d] decode the input data instead of encoding

Example for encoding:

```
wolfssl base64 -in plain_file.txt -out encoded_file.txt
```

Example for decoding:

```
wolfssl base64 -d -in encoded_file.txt -out decoded_file.txt
```