

JSTK Tools

*T*he author developed a number of simple command line tools and sample applications in the course of writing this book to explore various Java security capabilities and APIs. The discussion of these tools appears along with the relevant concepts throughout the book, with suggestion to look at the source code for complete working programs. This appendix provides comprehensive user level documentation on these tools at one place. Hopefully, these tools will prove to be useful to you in better understanding and utilizing the power of the security capabilities inherent in the Java platform.

These tools and applications have been packaged, along with source files and documentation, as an integrated toolkit and given a name – JSTK (**Java Security Tool Kit**). The directory structure of this package is shown in *Figure C-1*.

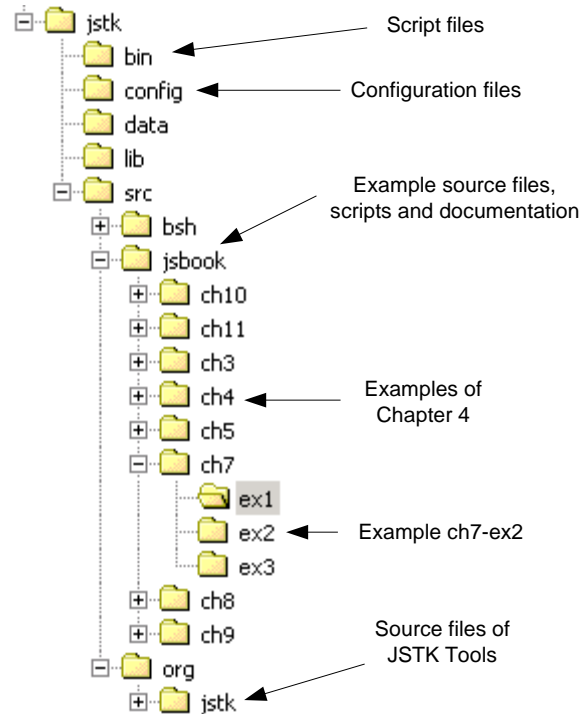


Figure C-1: JSTK installation directory structure

Follow the following steps to download, install and check successful installation of JSTK on a MS Windows machine.

1. **Prepare for Installation.** Make sure that you have J2SE SDK v1.4.x on your machine and the environment variable `JAVA_HOME` is pointing to the base installation directory.
2. **Get JSTK.** Download JSTK distribution file, `jstk-1_0.zip`, from <http://www.j2ee-security.net>, the companion website to this book.
3. **Install.** Unzip the distribution file. One way doing so is to issue command `"%JAVA_HOME%\bin\jar xvf jstk-1_0.zip"` in the directory where the downloaded file `jstk-1_0.zip` is saved. This should create the subdirectory `jstk-1.0` and place all source, binary and data files within the appropriate directory tree. This directory is referred to as the JSTK installation or home directory.
4. **Build.** This is an optional step and is required only if you modify one or more source files for the JSTK tools. To be able to compile the sources, you

must have Apache Ant installed on your machine and its `bin` directory in the `PATH`. To perform the build, simply issue the command `ant` from the JSTK installation directory. This command will create the jar file `jstk.jar` in the build subdirectory. To remove compiled classes and jar files, issue the command `ant clean`.

5. **Verify Installation.** Go to the JSTK installation directory and issue the command `bin\crypttool listp -csinfo`. This command should list all the available cryptographic service providers and their services.

Instructions for UNIX and Linux machines are very similar and can be obtained by performing following substitutions in the above instructions:

1. Replace `%JAVA_HOME%` by `$JAVA_HOME`.
2. Replace backslash by forward slash in the pathnames.
3. Add `.sh` to command file names. For example: `crypttool.sh`.

The rest of the appendix explains the individual JSTK tools and various commands and options supported by them. As you go through these tools, you will notice that almost all JSTK tools share the following structure: A JSTK tool takes a command and zero or more options as command line arguments. You can get a list of all the commands supported by the tool by specifying `help` command as in `bin\crypttool help`. Help on individual commands are obtained by placing `help` after the command as in `bin\crypttool listp help`.

It is best to invoke these tools from the JSTK installation directory, specifying the pathname of the script file.

CRYPTTOOL

NAME

crypttool – command line tool to explore and perform cryptographic operations.

SYNOPSIS

crypttool *command* (**help** | [*command-options*])

Executes `crypttool` with the specified *command*.

crypttool help

Displays all the commands available with `crypttool`.

crypttool *command* help

Displays all the *command-options* available with the *command*.

crypttool listp [*listp-options*]

Lists all the installed and configured cryptographic service providers.

crypttool listks [*listks-options*]
 Lists the entries in the specified keystore.

crypttool genk [*genk-options*]
 Generates a secret key.

crypttool genkp [*genkp-options*]
 Generates a public and private key pair.

crypttool crypt [*crypt-options*]
 Encrypts or decrypts the data of input file to an output file.

crypttool sign [*sign-options*]
 Creates or verifies a signature of data of a file.

crypttool digest [*digest-options*]
 Creates or verifies the digest of data in a file.

crypttool mac [*mac-options*]
 Creates or verifies message authentication code of data in a file.

crypttool bench [*bench-options*]
 Reports execution time of commands in a command file.

DESCRIPTION

The tool `crypttool` performs most of the cryptographic functions available in JCA and JCE. These functions include:

- Show available providers and information associated with each of the providers.
- Generate a secret key or a private and public key pair. A generated secret key can be (a) stored in a JCEKS keystore; (b) saved in a file; (c) printed on screen (hex value); or (d) discarded. The key saved in the file is essentially a serialized `SecretKey` object and hence not portable across providers. A private and public key pair can be (a) saved in a file; or (b) printed on screen. Similar to secret key, the public and private key pair is also a serialized object and not portable across providers.
- Encrypt and decrypt data using symmetric or asymmetric cryptography. Note that J2SE v1.4 doesn't support any asymmetric cipher.
- Create and verify digital signature. This operation involves asymmetric cryptography and requires a private and public key pair.
- Create and verify message digest.
- Create and verify Message Authentication Code (MAC).
- Measure performance of cryptographic operations.

Association of these operations with various `crypttool` commands is quite obvious.

OPTIONS

The table below lists all the different options supported by the utility **crypttool**. As not all options apply to every command, the applicable commands are also indicated. To get all the options supported by a command, issue the command: "**crypttool** *command* **help**"

-info	Display provider information. Applicable to <code>listp</code> command only.
-csinfo	Display cryptographic services available with each provider. Applicable to <code>listp</code> command only.
-props	Display properties set by each provider. Applicable to <code>listp</code> command only.
-provider <i>provider</i>	The provider to be used. Applicable for commands: <code>listks</code> , <code>genk</code> , <code>genkp</code> , <code>crypt</code> , <code>sign</code> , <code>mac</code> , <code>digest</code>
-keystore <i>keystore</i>	Keystore file. Default: <code>my.keystore</code> . Applicable for: <code>listks</code> , <code>genk</code> , <code>crypt</code> , <code>sign</code> , <code>mac</code>
-kstype <i>type</i>	Keystore type. Default: "JCEKS". Applicable for all commands that accept <code>-keystore</code> option.
-storepass <i>pass</i>	Keystore password. Default: "changeit". Applicable for all commands that accept <code>-keystore</code> option.
-alias <i>alias</i>	Alias to identify an entry in a keystore. Default: "mykey". Applicable for all commands that accept <code>-keystore</code> option.
-keypass <i>pass</i>	Password for a key entry. Default: none. Applicable for all commands that accept <code>-keystore</code> option.
-action <i>action</i>	Action on the generated key or key pair. Possible values: <code>print</code> , <code>store</code> , <code>save</code> , <code>discard</code> . Default: <code>discard</code> . Applicable to <code>genk</code> and <code>genkp</code> commands. Value <code>store</code> not supported for <code>genkp</code> .
-file <i>file</i>	File to save generated key or key pair. Applicable for commands <code>genk</code> and <code>genkp</code> .
-keyfile <i>file</i>	File to get the secret key or public and private key pair. This file must have been saved by <code>genk</code> or <code>genkp</code> command. Applicable to: <code>crypt</code> , <code>sign</code> , <code>mac</code>
-algorithm <i>alg</i>	Algorithm for the operation required for the command. Possible values depend on the operation and the provider. Applicable for: <code>genk</code> , <code>genkp</code> , <code>crypt</code> , <code>sign</code> , <code>mac</code> , <code>digest</code> .
-keysize <i>size</i>	Size of the key in bits. Possible values depend on the specified algorithm. Applicable to: <code>genk</code> and <code>genkp</code>
-op <i>op</i>	Operation to be performed with <code>crypt</code> command. Mandatory. No default. Possible values: <code>enc</code> , <code>dec</code> .

	Applicable to: <code>crypt</code> .
-infile <i>file</i>	File with input data. Mandatory. No default value.
	Applicable to: <code>crypt</code> , <code>sign</code> , <code>mac</code> , <code>digest</code>
-outfile <i>file</i>	File to save output data. Mandatory. No default value.
	Applicable to: <code>crypt</code> .
-password <i>pass</i>	Password for password based encryption or decryption. Mandatory for Password Based Encryption (PBE)
	Applicable to: <code>crypt</code>
-transform <i>trans</i>	Cipher transformation string in form <i>alg/mode/padding</i> . Default value: "DES/CFB8/NoPadding"
	Applicable to: <code>crypt</code> .
-iv <i>iv</i>	Initialization Vector. A string of 8 letters. Converted to byte array. Gets generated if not specified. Required based on the transform.
	Applicable to: <code>crypt</code>
-stream	Use Java <code>StreamCipher</code> API for encryption or decryption. Optional.
	Applicable to: <code>crypt</code> .
-verify	Verify the result of the operation indicated by command. Applicable to: <code>sign</code> , <code>mac</code> , <code>digest</code>
-sigfile <i>file</i>	File to save the signature bytes for <code>sign</code> command.
-sigbytes <i>bytes</i>	Hex data bytes of the signature. Could be used with <code>-verify</code> option in <code>sign</code> command to verify signature.
-mdfile <i>file</i>	File to save the digest bytes for <code>digest</code> command.
-mdbytes <i>bytes</i>	Hex data bytes of the digest. Could be used with <code>-verify</code> option in <code>digest</code> command to verify message digest.
-macfile <i>file</i>	File to save the MAC bytes for <code>mac</code> command.
-macbytes <i>bytes</i>	Hex data bytes of the MAC. Could be used with <code>-verify</code> option in <code>mac</code> command to verify MAC.
-cmdfile <i>file</i>	File with each command to be benchmarked. Sample command file: <code>%JSTK_HOME%\bin\ctbench.cmds</code> .
	Applicable to: <code>bench</code> .
-runcount <i>count</i>	How many runs for <code>bench</code> command?
-loopcount	How many iterations for each command within a run for the

<i>count</i>	bench command.
-warmuptime <i>time</i>	Warmup time in seconds for bench command. No. of iterations for running commands during this warmup phase is determined by measuring the time in running first iteration. So, the actual warmup time is usually less.
-showtime	Display execution time for a command.

EXAMPLES

```
crypttool listp -csinfo
```

Lists providers with details of cryptographic services supported by each provider. Very useful for exploring the services available with a Java platform.

```
crypttool genk -action store -keystore test.ks
```

Generates a DES (default algorithm) key of size 56 bits (default keysize) and stores it in a JCEKS (default keystore type) keystore file `test.ks` with keystore password "changeit" (default password) and the entry alias "mykey" (default alias).

```
crypttool listks -keystore test.ks
```

Lists the entries in the keystore file `test.ks`. Default keystore type "JCEKS" and password "changeit" is used.

```
crypttool crypt -op enc -infile build.xml \  
-outfile test.enc -keystore test.ks -iv 12345678
```

Encrypts file `build.xml` using the secret key in keystore `test.ks` and initialization vector as the byte array representation of string "12345678". The encrypted data is stored in the output file `test.enc`.

```
crypttool crypt -op dec -infile test.enc \  
-outfile test.dec -keystore test.ks -iv 12345678
```

Decrypts the file `test.enc` encrypted in last command using the same secret key. The decrypted data is stored in the output file `test.dec`.

```
crypttool mac -infile build.xml -keystore test.ks \  
-macfile test.mac  
crypttool mac -infile build.xml -keystore test.ks \  
-macfile test.mac -verify
```

Computes the MAC of the input file `build.xml` and verifies it. The secret key of earlier operations is used here as well.

```
crypttool genkp -action save -file test.kp \  
-algorithm RSA
```

Generates RSA key pair of keysize 512 (default keysize) and saves the serialized KeyPair object to the file `test.kp`.

```
crypttool sign -infile build.xml -sigfile test.sig \  
-keyfile test.kp -algorithm SHA1WithRSA
```

Signs the file `build.xml` with the RSA private key using SHA1WithRSA algorithm and saves the signature in the file `test.sig`.

```
crypttool sign -infile build.xml -sigfile test.sig \  
-keyfile test.kp -algorithm SHA1WithRSA -verify
```

Verifies the signature created by the last command.

CERTTOOL

NAME

certtool – Command line tool to setup a simple CA and issue, show, revoke and verify certificates.

SYNOPSIS

```
certtool command (help | [command-options])
```

Executes `certtool` with the specified `command`.

```
certtool help
```

Displays all the commands available with `certtool`.

```
certtool command help
```

Displays all the `command-options` available with the `command`.

```
certtool setupca [setupca-options]
```

Sets up a file based simple CA..

```
certtool issue [issue-options]
```

Issues a certificate and updates the CA files.

```
certtool show [show-options]
```

Displays the contents of a certificate, certification path or CRL.

```
certtool revoke [revoke-options]
```

Revokes a previously issued certificate.

```
certtool crl [crl-options]
```

Generates a CRL file of all the revoked certificates.

```
certtool validate [validate-options]
```

Validates a certificate.

DESCRIPTION

The tool **certtool** is a command line utility to set up a minimal CA. During setup, it can either generate a self-signed certificate or use a certificate signed by another CA. After setup, **certtool** can be used to issue signed certificates taking a CSR as input, revoke a previously issued certificate, generate a CRL (**Certificate Revocation List**) and so on. All information related to the **certtool**-based CA is stored in flat files within a directory tree rooted at the directory specified during the setup.

OPTIONS for **certtool setupca**

-cadir <i>dir</i>	Directory to store internal data. Default: <i>cadir</i> .
-dn <i>dn</i>	Distinguished Name of the CA. Default:[CN=JSTK Test Root CA, OU=JSTK Operations, O=JSTK Inc, C=US].
-capath <i>pathlen</i>	Maximum permissible depth of the CA hierarchy rooted at this CA. Default: 2.
-serial <i>serialno</i>	Serial no. the CA certificate. Default: 100
-keyalg <i>alg</i>	Algorithm for key-pair generation. Default: RSA. Other possible value is DSA.
-keysize <i>keysz</i>	Key size in bits. Default: 2048.
-sigalg <i>sigalg</i>	Signature algorithm. Should match the key algorithm. Default: SHA1withRSA.
-password <i>passwd</i>	Password for CA keystore. This is mandatory and there is no default for it.

OPTIONS for **certtool issue**

-cadir <i>dir</i>	certtool CA directory. Default: <i>cadir</i> .
-ca	Flag to indicate that the issued certificate is a CA certificate
-capath <i>pathlen</i>	Maximum permissible depth of the CA hierarchy rooted at this CA. Default: 0.
-csrfile <i>csrfile</i>	Input file with the Certificate Signing Request.
-cerfile <i>cerfile</i>	Output file to store the certificate.
-cpfmt <i>cpfmt</i>	Certification path format for the output file. Default: PKCS7. Other possible values are PKIPATH and X509.
-keyalg <i>alg</i>	Algorithm for key-pair generation. Default: RSA. Other possible value is DSA.
-keysize <i>keysz</i>	Key size in bits. Default: 2048.

-sigalg <i>sigalg</i>	Signature algorithm. Should match the key algorithm. Default: SHA1WithRSA.
-password <i>passwd</i>	Password specified at the time of CA setup. This is mandatory and there is no default for it.

OPTIONS for `certtool revoke`

-cadir <i>dir</i>	<code>certtool</code> CA directory. Default: <code>cadir</code> .
-cerfile <i>cerfile</i>	input file having the certificate to be revoked.
-password <i>passwd</i>	Password specified at the time of CA setup. This is mandatory and there is no default for it.

OPTIONS for `certtool crl`

-cadir <i>dir</i>	<code>certtool</code> CA directory. Default: <code>cadir</code> .
-crlfile <i>crlfile</i>	Output file to store the CRL of all the revoked certificates.
-password <i>passwd</i>	Password specified at the time of CA setup. This is mandatory and there is no default for it.

OPTIONS for `certtool show`

-infile <i>infile</i>	Input file.
------------------------------	-------------

EXAMPLES

```
certtool setupca -password changeit
```

Sets up the files for a simple file-based CA. Directory `cadir` is created to hold all the files and sub-directories for maintaining information about the CA. The self signed certificate for the CA and its private key are stored in keystore `cadir\ca.ks`, protected by password `changeit` and within `cakey` entry.

```
keytool -genkey -keystore test.ks -storepass changeit
keytool -certreq -file test.csr -keystore test.ks \
-storepass changeit
certtool issue -csrfile test.csr -password hello
```

The first `keytool` command creates keystore `test.ks` with a self-signed certificate for the identity information supplied. The second `keytool` command generates a CSR from this self-signed certificate. This CSR is used to issue a CA signed certificate by the utility `certtool`. The issued certificate is stored in file `my.cer`.

```
certtool show -infile my.cer
```

Displays the contents of the issued certificate.

```
certtool setupca -cadir cadir1 -password hello  
keytool -certreq -file ca1.csr -keystore cadir1\ca.ks \  
  -storepass hello -alias cakey -storetype JCEKS  
certtool issue -csrfile ca1.csr -cerfile ca1.cer \  
  -password hello  
keytool -import -file ca1.cer -keystore cadir1\ca.ks \  
  -storepass hello -alias cakey -storetype JCEKS
```

Creates a sub-CA in subdirectory `cadir1`. The basic mechanism to setup a CA with CA directory `cadir1`: generate a CSR from its keystore, issue a certificate as per this CSR using the super-CA and then import the issued certificate to the original keystore.

SSLSETUP

NAME

sslsetup – command line tool to setup keystore and environment for SSL communication.

SYNOPSIS

sslsetup ss-certs

Creates keystore and truststore for client and server programs with self-signed certificates.

sslsetup cs-certs

Creates keystore and truststore for client and server programs with CA signed certificates. The assumption is that a CA has been setup using the JSTK tool **certtool**.

sslsetup server-env

Sets environment variable `JSTK_OPTS` so that appropriate system properties are passed to the JVM on invoking "**ssltool server**" command.

sslsetup client-env

Sets environment variable `JSTK_OPTS` so that appropriate system properties are passed to the JVM on invoking "**ssltool client**" command.

DESCRIPTION

The tool **sslsetup** is a simple script to automate long sequence of **keytool** and **certtool** commands to create keystore and truststore files for client and server programs; and to set environment variable `JSTK_OPTS` with proper system property definitions. In this regard, **sslsetup** is nothing but a convenient shortcut to save typing. Look at the script file in bin directory of JSTK distribution for what it really does under the hood.

Files created by “`sslsetup ss-certs`” or “`sslsetup cs-certs`” command:

`server.ks`: Stores the server’s certificate with the corresponding private key.

`client.ks`: Stores the client’s certificate with the corresponding private key.

`server.ts`: Stores the client’s or issuer’s (in case of CA signed) certificate.

`client.ts`: Stores the server’s or issuer’s (in case of CA signed) certificate.

All files are JCEKS type keystore files with password `changeit`.

Value of `JSTK_OPTS` set by “`sslsetup server-env`” command:

```
-Djavax.net.ssl.keyStore=server.ks -Djavax.net.ssl.keyStoreType=JCEKS \
-Djavax.net.ssl.keyStorePassword=changeit -Djavax.net.ssl.trustStore \
=server.ts -Djavax.net.ssl.trustStoreType=JCEKS
```

Value of `JSTK_OPTS` set by “`sslsetup server-env`” command:

```
-Djavax.net.ssl.keyStore=client.ks -Djavax.net.ssl.keyStoreType=JCEKS \
-Djavax.net.ssl.keyStorePassword=changeit -Djavax.net.ssl.trustStore \
=client.ts -Djavax.net.ssl.trustStoreType=JCEKS
```

KNOWN BUGS/LIMITATIONS

1. It is not possible to specify the signature algorithm (RSA or DSA) and keysize during certificate generation.

SSLTOOL

NAME

`ssltool` – command line tool to explore SSL support in Java (JSSE)

SYNOPSIS

```
ssltool command (help | [command-options])
```

Executes `ssltool` with the specified *command*.

```
ssltool help
```

Displays all the commands available with `ssltool`.

```
ssltool command help
```

Displays all the *command-options* available with the *command*.

```
ssltool show -cs
```

Shows all the supported and enabled cipher suites.

```
ssltool server [server-options]
```

Runs `ssltool` as a server program.

```
ssltool client [client-options]
```

Runs `ssltool` as a client program.
ssltool proxy [*proxy-options*]
 Runs `ssltool` as a proxy (tunnel) program.

DESCRIPTION

The tool **ssltool** is a utility program to explore SSL support in the Java platform. It does so by running as a server, a client or a proxy program and by querying the Java platform on supported SSL protocols versions and cipher suites.

While running as server, it serves requests as per the incoming protocol, as specified by the option **-inproto**. When the incoming protocol is either **TCP** or **SSL** then the server listens for incoming connections. Once a connection is accepted, it spawns a thread to service messages on that connection. Action taken on receiving data bytes depends on the value of the options **-mode** and **-action**. For example, in the **echo** mode the server writes back whatever it receives to the same connection. In the **bench** mode, it can discard the received data (**read-only** action); write back the received data (**read-write** action) or simply wait for the connection to get closed (**accept-wait** action). Other supported incoming protocols are **HTTP**, **HTTPS**, **RMI** and **SRMI (RMI over SSL)**. With **HTTP** and **HTTPS**, it simply returns the document specified by the requested URL and the value of **-action** is ignored. With **RMI** and **SRMI**, it runs an in-process **rmiregistry** and an **RMI** server class.

The client role is complementary to the server role, with the outgoing protocols (option **-outproto**) the same as incoming protocols supported by the server. With **TCP** and **SSL** protocols and **echo** mode, the client prompts the user to type a message, terminated by a new line, and sends the message to the server over the connection established during client initialization. With **RMI** and **SRMI** protocols, the message supplied as a byte array argument to the method call. In **bench** mode, the client writes **bufsize** bytes (**write-only** action); writes and reads **bufsize** bytes (**write-read** action) or simply opens and closes connections (**open-close** action) in a loop with the loop count specified by the option **-num**.

A proxy between a client and server can be used to analyze the TCP messages being exchanged. **ssltool** based proxy operates at TCP level, so it can work with any client and server program. It waits for TCP connections at the port specified by the option **-inport** and forwards the connection to the target address specified by options **-host** and **-port**. The databytes exchanged can be analyzed by specifying one or more supported protocol analyzers. Currently, only two analyzers are supported: **dd** (data display) and **ssl**. The former simply displays the data exchanged as hex bytes and the later parses the data as per SSL message definitions. Protocol analyzers are specified by option **-patype**. More than one analyzer can be specified by supplying a comma separated list as in **-patype "dd,ssl"**.

Essentially, **ssltool** enables you to

- Query the Java platform for supported protocols and cipher suites.

- Establish SSL connection and exchange data between any two machines connected by a TCP/IP network with the specified authentication and trusted certificates.
- Perform HTTPS communication.
- Perform RMI over SSL.
- Benchmark SSL performance and compare it with TCP performance.
- Analyze SSL protocol messages between any pair of SSL client and server.

OPTIONS for `ssltool` server

-inport <i>portno</i>	TCP port to accept incoming connection. Default: 9000. Valid for -inproto values of TCP, SSL, HTTP and HTTPS only. For -inproto values of RMI and SRMI, this is the port associated with the in-process rmi registry.
-inproto <i>proto</i>	Protocol to accept and service requests. Valid values: TCP, SSL, HTTP, HTTPS, RMI and SRMI. Default: TCP. For protocols SSL, HTTPS and SRMI, system properties for keystore and truststore can be passed to the JVM by setting environment variable <code>JSTK_OPTS</code> .
-inetaddr <i>addr</i>	IP address of the network interface card to be used for TCP and SSL communication. Default: none.
-mode <i>mode</i>	Mode to service the requests. Valid values: echo and bench. Default: echo.
-action <i>action</i>	Valid for protocols TCP, SSL, RMI and SRMI in bench mode only. Possible values: read-only: discard the data. read-write: send back the data to the client. accept-wait: for connection accept and wait for closing by the client. Not applicable to RMI and SRMI. Default: read-only
-bufsize <i>size</i>	Size of the buffer (in bytes) to read data. Default: 8192
-needcauth	Flag to indicate mandatory client authentication. Applicable to SSL, HTTPS and SRMI protocols only.
-wantcauth	Flag to indicate negotiation for client authentication. Applicable to SSL, HTTPS and SRMI protocols only.
-csfile <i>file-name</i>	File to read cipher suits to be enabled. The file with the specified filename contains the cipher suite symbolic names, one per line. Applicable to SSL, HTTPS and SRMI protocols only.

-nio	Flag to indicate use of NIO buffers and socket calls for TCP based communication.
-verbose	Display execution status. Helpful for debugging.

OPTIONS for `ssltool client`

-port <i>portno</i>	TCP port to make outgoing connection. Default: 9000. Valid for -outproto values of TCP and SSL only. For -outproto values of RMI and SRMI, this is the port used for RMI registry lookup.
-host <i>hostname</i>	Hostname or IP address of the machine running the server program. Default: localhost. Valid for -outproto values of TCP and SSL only. For -outproto values of RMI and SRMI, this is the host used for RMI registry lookup.
-outproto <i>proto</i>	Protocol to make requests. Valid values: TCP, SSL, RMI and SRMI. Default: TCP. For protocols SSL and SRMI, system properties for keystore and truststore can be passed to the JVM by setting environment variable <code>JSTK_OPTS</code> .
-inetaddr <i>addr</i>	IP address of the network interface card to be used for TCP and SSL communication. Default: none.
-mode <i>mode</i>	Mode to make the requests. Valid values: echo and bench. Default: echo.
-action <i>action</i>	Valid for protocols TCP, SSL, RMI and SRMI in bench mode only. Possible values: write-only: write but do not read. write-read: write and read. open-close: open and close connections. Not applicable to RMI and SRMI. Default: write-only
-invalidate	Invalidate the SSLContext associated with the SSL connection. Used for benchmarking SSL connection setup overhead (<code>-outproto ssl -mode bench -action open-close</code>).
-bufsize <i>size</i>	Size of the buffer (in bytes) to write data in bench mode. Default: 8192
-num <i>num</i>	Loop count in bench mode. Default: 2048.
-url <i>url</i>	http or https URL to access. Other options such as <code>-host</code> , <code>-port</code> , <code>-outproto</code> , <code>-action</code> , <code>-nio</code> etc. are ignored.

-nio	Flag to indicate use of NIO buffers and socket calls for TCP based communication.
-csfile <i>file-name</i>	File to read cipher suits to be enabled. The file with the specified filename contains the cipher suite symbolic names, one per line. Applicable to SSL, HTTPS and SRMI protocols only.
-verbose	Display execution status. Helpful for debugging.

OPTIONS for **ssltool proxy**

-inport <i>portno</i>	TCP port to accept incoming connection. Default: 8995.
-port <i>portno</i>	TCP port to make outgoing connection. Default: 9000.
-host <i>hostname</i>	Hostname or IP address of the target. Default: localhost.
-bufsize <i>size</i>	Size of the buffer (in bytes) to read data. Default: 8192
-patype <i>palist</i>	Protocol Analyzer types to analyze traffic. Default: none. Valid values (can also specify a comma separated list): dd: display data as hex bytes. ssl: parse SSL record headers and handshake messages. dd, ssl or ssl, dd
-nio	Flag to indicate use of NIO buffers and socket calls for TCP based communication.
-verbose	Display execution status. Helpful for debugging.

EXAMPLES

ssltool server

Runs a server listening for TCP connections on port 9000. Displays information about accepted connections and received bytes. Writes back the received data to the connection. Enter **Ctrl-C** to terminate the program.

ssltool client

Runs a client program that establishes a TCP connection to the server running on the same machine and listening for connection at port 9000. Prompts the user to enter a message. Reads the message, sends it to the server, reads the response and prints it on the screen. Enter **quit** at the prompt to exit the client.

ssltool proxy -patype dd

Runs a proxy program that tunnels the connections targeted to port 8995 to the port 9000 on the same machine. Displays all the exchanged bytes in hex. You should run the client by issuing command "**ssltool client -port 8995**" to connect to proxy in place of the server in the previous example. Enter **Ctrl-C** to terminate the program.

```

sslsetup ss-certs
sslsetup server-env
ssltool server -inproto ssl

```

This sequence of commands creates keystore and truststore files, `server.ks`, `client.ks`, `server.ts` and `client.ts` in the current directory, populated with self-signed certificates for both client and server; sets proper values to the environment variable `JSTK_OPTS` and runs the server program to accept SSL connections.

```

sslsetup client-env
ssltool client -outproto ssl

```

Sets up the `JSTK_OPTS` environment variable for running the client (assuming that the commands are executed on the same machine and from the same directory as the previous sequence of commands) and runs the client to establish SSL connection with the server.

```

ssltool server -inproto ssl -mode bench -action accept-
wait -csfile cs.txt

```

Runs the server for benchmarking SSL connection setup overhead using the cipher suite listed in file `cs.txt`. Assumes that the `JSTK_OPTS` is set properly to accept SSL connections.

```

ssltool client -outproto ssl -mode bench -action open-
close -host venus -csfile cs.txt -invalidate

```

Runs the client for benchmarking SSL connection setup overhead to the server running on host `venus` using the cipher suite listed in file `cs.txt`. Assumes that the `JSTK_OPTS` is set properly to accept SSL connections.

ASN1PARSE – PARSER FOR DER OR PEM ENCODED CONTENT

NAME

asn1parse – tool to parse and display ASN.1 fields of a DER or PEM encoded file.

SYNOPSIS

```
asn1parse file [-encode encoded-file]
```

DESCRIPTION

This tool is for verifying and displaying the encoding of DER or PEM encoded files. Useful for debugging. This tool first checks if the input file is in PEM format by examining the header and footer. If so, it converts it into equivalent binary file by applying base64 de-

coding on the body, removing the header and footer. The resulting file is parsed as per the DER (Distinguished Encoding Rules) and the successfully parsed elements are displayed on the screen.

With **-encode** option, it writes the parsed content into the specified file in DER format. This option can be used to convert a PEM format file into a DER format.

EXAMPLES

asn1parse server.cer

Displays the ASN.1 fields of the DER encoded certificate file `server.cer`.

asn1parse server.pem -encode server.cer

Displays the ASN.1 fields of the PEM encoded file `server.pem` and writes the content in DER format to the `server.cer` file.