SSL tests

SSL testcases are configured in the `ssl-tests` directory.

Each `ssl_*.cnf.in` file contains a number of test configurations. These files are used to generate testcases in the OpenSSL CONF format.

The precise test output can be dependent on the library configuration. The test harness generates the output files on the fly.

However, for verification, we also include checked-in configuration outputs corresponding to the default configuration. These testcases live in `test/ssl-tests/*.cnf` files.

For more details, see `ssl-tests/01-simple.cnf.in` for an example.

Configuring the test

First, give your test a name. The names do not have to be unique.

An example test input looks like this:

```
{
    name => "test-default",
    server => { "CipherString" => "DEFAULT" },
    client => { "CipherString" => "DEFAULT" },
    test   => { "ExpectedResult" => "Success" },
}
```

The test section supports the following options

Test mode

- Method - the method to test. One of DTLS or TLS.

- HandshakeMode - which handshake flavour to test:

  - Simple - plain handshake (default)
  - Resume - test resumption
  - RenegotiateServer - test server initiated renegotiation
  - RenegotiateClient - test client initiated renegotiation

When HandshakeMode is Resume or Renegotiate, the original handshake is expected to succeed. All configured test expectations are verified against the second handshake.

- ApplicationData - amount of application data bytes to send (integer, defaults to 256 bytes). Applies to both client and server. Application data is sent in 64kB chunks (but limited by MaxFragmentSize and available parallelization, see below).

- MaxFragmentSize - maximum send fragment size (integer, defaults to 512 in tests - see `SSL_CTX_set_max_send_fragment` for documentation). Applies to both client and server. Lowering the fragment size will split handshake and application data up between more `SSL_write` calls, thus allowing to exercise different code paths. In particular, if the buffer size (64kB) is at least four times as large as the maximum fragment, interleaved multi-buffer crypto implementations may be used on some platforms.

Test expectations

- ExpectedResult - expected handshake outcome. One of
  - Success - handshake success
  - ServerFail - serverside handshake failure
  - ClientFail - clientside handshake failure
  - InternalError - some other error
- ExpectedClientAlert, ExpectedServerAlert - expected alert. See `test/helpers/ssl_test_ctx.c` for known values. Note: the expected alert is currently matched against the *last* received alert (i.e., a fatal alert or a `close_notify`). Warning alert expectations are not yet supported. (A warning alert will not be correctly matched, if followed by a `close_notify` or another alert.)
- ExpectedProtocol - expected negotiated protocol. One of SSLv3, TLSv1, TLSv1.1, TLSv1.2.
- SessionTicketExpected - whether or not a session ticket is expected
  - Ignore - do not check for a session ticket (default)
  - Yes - a session ticket is expected
  - No - a session ticket is not expected
- SessionIdExpected - whether or not a session id is expected
  - Ignore - do not check for a session id (default)
  - Yes - a session id is expected
  - No - a session id is not expected
- ResumptionExpected - whether or not resumption is expected (Resume mode only)
  - Yes - resumed handshake
  - No - full handshake (default)
- ExpectedNPNProtocol, ExpectedALPNProtocol - NPN and ALPN expectations.
- ExpectedTmpKeyType - the expected algorithm or curve of server temp key
- ExpectedServerCertType, ExpectedClientCertType - the expected algorithm or curve of server or client certificate
- ExpectedServerSignHash, ExpectedClientSignHash - the expected signing hash used by server or client certificate
- ExpectedServerSignType, ExpectedClientSignType - the expected signature type used by server or client when signing messages
- ExpectedClientCANames - for client auth list of CA names the server must send. If this is "empty" the list is expected to be empty otherwise it is a file of certificates whose subject names form the list.
- ExpectedServerCANames - list of CA names the client must send, TLS 1.3 only. If this is "empty" the list is expected to be empty otherwise it is a file of certificates whose subject names form the list.

Configuring the client and server

The client and server configurations can be any valid `SSL_CTX` configurations. For details, see the manpages for `SSL_CONF_cmd`.

Give your configurations as a dictionary of CONF commands, e.g.

```
server => {
    "CipherString" => "DEFAULT",
    "MinProtocol" => "TLSv1",
}
```

The following sections may optionally be defined:

- server2 - this section configures a secondary context that is selected via the ServerName test option. This context is used whenever a ServerNameCallback is specified. If the server2 section is not present, then the configuration matches server.
- resume_server - this section configures the client to resume its session against a different server. This context is used whenever HandshakeMode is Resume. If the resume_server section is not present, then the configuration matches server.
- resume_client - this section configures the client to resume its session with a different configuration. In practice this may occur when, for example, upgraded clients reuse sessions persisted on disk. This context is used whenever HandshakeMode is Resume. If the resume_client section is not present, then the configuration matches client.

Configuring callbacks and additional options

Additional handshake settings can be configured in the `extra` section of each client and server:

```
client => {
    "CipherString" => "DEFAULT",
    extra => {
        "ServerName" => "server2",
    }
}
```

Supported client-side options

- ClientVerifyCallback - the client's custom certificate verify callback. Used to test callback behaviour. One of
    - None - no custom callback (default)
    - AcceptAll - accepts all certificates.
    - RejectAll - rejects all certificates.
- ServerName - the server the client should attempt to connect to. One of
    - None - do not use SNI (default)
    - server1 - the initial context
    - server2 - the secondary context
    - invalid - an unknown context
- CTValidation - Certificate Transparency validation strategy. One of
    - None - no validation (default)
    - Permissive - SSL_CT_VALIDATION_PERMISSIVE
    - Strict - SSL_CT_VALIDATION_STRICT

Supported server-side options

- ServerNameCallback - the SNI switching callback to use

    - None - no callback (default)
    - IgnoreMismatch - continue the handshake on SNI mismatch
    - RejectMismatch - abort the handshake on SNI mismatch

- BrokenSessionTicket - a special test case where the session ticket callback does not initialize crypto.

    - No (default)
    - Yes

Mutually supported options

- NPNProtocols, ALPNProtocols - NPN and ALPN settings. Server and client protocols can be specified as a comma-separated list, and a callback with the recommended behaviour will be installed automatically.

- SRPUser, SRPPassword - SRP settings. For client, this is the SRP user to connect as; for server, this is a known SRP user.

Default server and client configurations

The default server certificate and CA files are added to the configurations automatically. Server certificate verification is requested by default.

You can override these options by redefining them:

```
client => {
    "VerifyCAFile" => "/path/to/custom/file"
}
```

or by deleting them

```
client => {
    "VerifyCAFile" => undef
}
```

Adding a test to the test harness

1. Add a new test configuration to `test/ssl-tests`, following the examples of existing `*.cnf.in` files (for example, `01-simple.cnf.in`).

2. Generate the generated `*.cnf` test input file. You can do so by running `generate_ssl_tests.pl`:

   $ ./config $ cd test $ TOP=.. perl -I ../util/perl/ generate_ssl_tests.pl
   ssl-tests/my.cnf.in default > ssl-tests/my.cnf

where `my.cnf.in` is your test input file and `default` is the provider to use. For all the pre-generated test files you should use the default provider.

For example, to generate the test cases in `ssl-tests/01-simple.cnf.in`, do

```
$ TOP=.. perl -I ../util/perl/ generate_ssl_tests.pl \
  ssl-tests/01-simple.cnf.in default > ssl-tests/01-simple.cnf
```

Alternatively (hackish but simple), you can comment out

```
unlink glob $tmp_file;
```

in `test/recipes/80-test_ssl_new.t` and run

```
$ make TESTS=test_ssl_new test
```

This will save the generated output in a `*.tmp` file in the build directory.

3. Update the number of tests planned in `test/recipes/80-test_ssl_new.t`. If the test suite has any skip conditions, update those too (see `test/recipes/80-test_ssl_new.t` for details).

Running the tests with the test harness

```
HARNESS_VERBOSE=yes make TESTS=test_ssl_new test
```

Running a test manually

These steps are only needed during development. End users should run `make test` or follow the instructions above to run the SSL test suite.

To run an SSL test manually from the command line, the `TEST_CERTS_DIR` environment variable to point to the location of the certs. E.g., from the root OpenSSL directory, do

```
$ CTLOG_FILE=test/ct/log_list.cnf TEST_CERTS_DIR=test/certs test/ssl_test \
  test/ssl-tests/01-simple.cnf default
```

or for shared builds

```
$ CTLOG_FILE=test/ct/log_list.cnf  TEST_CERTS_DIR=test/certs \
  util/wrap.pl test/ssl_test test/ssl-tests/01-simple.cnf default
```

In the above examples, `default` is the provider to use.

Note that the test expectations sometimes depend on the Configure settings. For example, the negotiated protocol depends on the set of available (enabled) protocols: a build with `enable-ssl3` has different test expectations than a build with `no-ssl3`.

The Perl test harness automatically generates expected outputs, so users who just run `make test` do not need any extra steps.

However, when running a test manually, keep in mind that the repository version of the generated `test/ssl-tests/*.cnf` correspond to expected outputs in with the default Configure options. To run `ssl_test` manually from the command line in a build with a different configuration, you may need to generate the right `*.cnf` file from the `*.cnf.in` input first.