TLS Documentation

Release 0.0

Individual Contributors

July 19, 2016

Contents

CHAPTER 1

Python TLS API

class `ClientTLS`(*server_hostname*, *cipher_suites*, *trust_root=DEFAULT*, *client_certificate_store=None*)

Parameters

- **server_hostname** (*bytes*) – The hostname of the server that will be connected to.
- **cipher_suites** (*list*) – The list of supported cipher suites.
- **trust_root** (TrustStore) – The trust root.
- **client_certificate_store** (ClientCertificateStore) – The certificate that the client will present to the server.

start(*write_to_wire_callback*, *wire_close_callback*, *verify_callback=None*)

Parameters

- **write_to_wire_callback** (*callable*) – Callable of one argument of bytes type. It will be called when TLS data should be sent over the transport.
- **wire_close_callback** (*callable*) – Callable of one argument of bool type, called immediate. It will be called when the TLS protocols mandates a transport shutdown. The read side of the connection must always be shut down immediately and no further data should be delivered to the connection. If immediate is True, then the transport should close the write side of the transport and free all associated resources as soon as possible. If immediate is False, then the transport should make a reasonable attempt to deliver the bytes already sent to write_to_wire_callback (which will be a close_alert message), meaning it can wait for a configured timeout before closing down the write side of the connection.
- **verify_callback** (*callable*) – Callable of two arguments: a list of *Certificate* objects, and a *Connection* object. It will be called once per negotiation with a list of Certificates and the connection object. The certificates are in chain order, starting with the leaf certificate and ending with the root-most certificate. Specifying a verify_callback does *not* override the basic verification that PyTLS does, such as certificate chain validation, basic certificate checks, and hostname validation. verify_callback has no particular contract; return values will be ignored. If any exception is raised, the connection will be invalidated and any future calls to *Connection.data_from_wire()* or *Connection.data_from_application()* will raise *InvalidatedError*. It's up to the user to decide what to do during verification, such as invoking *Connection.send_alert()* or simply closing the connection.

Return Connection the client connection.

Start a TLS connection. The write_to_wire_callback will be invoked with the initial data for TLS negotiation.

class ServerTLS(*certificates*, *dh_params=None*)

Parameters

- **certificates** (*ServerCertificates*) – A collection of server certificates, usually an instance of either *ServerCertificateChain* or *SNIServerCertificates*.
- **dh_params** (*bytes*) – Optional diffie-hellman parameters in DER format.

start(*write_to_wire_callback*, *verify_callback=None*)

Return Connection the server connection.

See ClientTLS.start.

class Connection

data_from_wire(*input*)

Parameters **input** (*bytes*) – Data that was received from some low-level transport and should be processed by the TLS implementation.

Return bytes Any application data that was in the input.

Raises

- ***TLSAlertError*** – When certain TLS Alert messages occur in the input.
- ***BadTLSDataError*** – When the input data was somehow invalid, such as when decryption failed or the protocol was not followed.
- ***InvalidatedError*** – When the connection has been invalidated due to a previous error and will accept no further data.

Given data read from a transport, invoke any callbacks for e.g. connection negotiation or heartbeats, etc, and return decrypted application data, if any. If the input data is somehow invalid, a TLS Alert message will be passed to the write callback, and a BadTLSDataError will be raised. In certain cases of receipt of invalid data, after (sometimes) sending a TLS Alert, this connection will be invalidated such that data_from_wire and data_from_application will raise *InvalidatedError*. Note that any incomplete data in the input may be buffered by the implementation until further calls to data_from_wire complete the messages.

data_from_application(*output*)

Parameters **output** (*bytes*) – Application data to encrypt and send over the transport.

Raises ***InvalidatedError*** – When the connection has been invalidated due to a previous error.

Given plaintext application data, invoke the write callback with the encrypted data.

send_alert(*alert_code*, *level=None*)

Parameters

- **alert_code** – The alert code to send in a TLS Alert message. Must be one of the constants specified in this module (TBD).
- **level** – Must be ALERT_WARNING or ALERT_FATAL. If not specified, a default will be specified based on alert_code if the TLS specification mandates a particular level for the code.

Raises *InvalidAlertLevel* – When an alert_code is passed that is incompatible with the passed level.

Invoke the write callback with a TLS alert message. Usually this is invoked automatically by a method like data_from_wire, but it may be useful to call this in your verify_callback. If the level is passed, the alert code *must* be compatible according to the TLS spec, otherwise *InvalidAlertLevel* will be raised. If the level is not passed and the alert code is ambiguous according to the spec, *InvalidAlertLevel* will also be raised in this case. Certain send_alert() calls may invalidate the connection, in which case further calls to data_from_application and data_from_wire will fail with *InvalidatedError*.

application_finished()

Indicate that the application is finished sending data to `data_from_application`. If the connection has already started, this will invoke the write callback with a TLS Finished message.

CHAPTER 2

Certificate APIs

Definition: a "leaf" certificate is a non-CA certificate.

class Certificate

get_asn1_bytes()
Get the ASN1-format bytes of the certificate.

class ClientCertificateStore

get_certificate_chain_for_roots(*roots*, *certificate_chain_callback*)

Parameters

- **roots** (*set*) – A set of keyless certificate that the server specified as the valid roots that a client certificate must chain to.
- **certificate_chain_callback** (*callable*) – The callback that this method should eventually invoke to specify the client certificates to send.

This method is intended to be implemented by the user, NOT called by the user.

Get the client certificate chain to send to the server, based on the roots specified by the server. The result should be specified by calling certificate_chain_callback. It must be passed either a single certificate chain (with ONE leaf certificate that MUST have a private key), or None to indicate no client certificates are available.

The certificates must chain to one of the roots specified by the server, or *NoCertificateChainError* will be raised. Invoking this callback more than once will result in *InvalidatedError* being raised. The callback may also raise *LeafCertificateHasNoPrivateKeyError*, *MoreThanOneLeafCertificateError*, or *NoLeafCertificateError*.

get_default_certificate_chain(*certificate_chain_callback*)

Parameters **certificate_chain_callback** (*callable*) – The callback that this method should eventually invoke to specify the client certificates to send.

This method is intended to be implemented by the user, NOT called by the user.

Get the default client certificate in the case that the server did not provide roots that the client certificate must chain to. The result should be specified by calling certificate_chain_callback. It must be passed either a single certificate chain (with ONE leaf certificate that MUST have a private key), or None to indicate no client certificates are available.

Invoking this callback more than once will result in *InvalidatedError* being raised. The callback may also raise *LeafCertificateHasNoPrivateKeyError*, *MoreThanOneLeafCertificateError*, or *NoLeafCertificateError*.

class TrustStore(*certificates*)

Parameters **certificates** (*set*) – A set of Certificate objects, none of which may have private keys.

Create a store of trusted CA certificates to be used with ClientTLS. No methods are public. If any private keys are found in any of the certificates, *ExtraneousPrivateKeyError* will be raised.

class ServerCertificates

An abstract base class representing the type of operations possible on a collection of server certificates.

get_certificate_chain_for_server_name(*server_name*, *certificate_chain_callback*)

Parameters

- **server_name** (*bytes*) – The server name.
- **certificate_chain_callback** (*callable*) – A callable of one argument that must be eventually called by this method.

This method is intended to be implemented by the user, NOT called by the user.

Get the server chain to send to the client when the client is using Server Name Indication (SNI). Implement this method to invoke the certificate_chain_callback with a collection of certificates with ONE leaf certificate that MUST have a private key. None may be passed to the certificate_chain_callback in case no certificates can be found, in which case a TLS Alert will be sent. Passing a "default" certificate chain that doesn't match the server name is acceptable.

Invoking this callback more than once will result in *InvalidatedError* being raised. The callback may be invoked at any point after this method is invoked; it needn't be invoked synchronously. The callback may also raise *LeafCertificateHasNoPrivateKeyError*, or *NoLeafCertificateError*.

class ServerCertificateChain(*chain*)

provides ServerCertificates

Parameters **chain** (*set*) – A single chain of certificates, the leaf of which MUSt have a private key.

Specify the certificate chain that will be sent to all clients.

class SNIServerCertificates(*certificates*, *default=set()*)

provides ServerCertificates

Parameters

- **certificates** (*set*) – A set of certificates that may contain multiple distinct certificate chains. Any leaf certificates MUST have private keys.
- **default** (*set*) – A single certificate chain, the leaf of which MUST have a private key.

Represents a SNI-capable set of certificates for use with ServerTLS.

CHAPTER 3

Exceptions

class TLSAlertError

Attribute alert_code code of the alert

Attribute alert_level level of the alert

Raised when a TLS Alert message was received from the peer.

class BadTLSDataError
Raised when invalid TLS data was received from the peer.

class InvalidatedError
Raised when it's no longer valid to call a method or callback based on previous state. e.g., a certificate_chain_callback from *ServerCertificates.get_certificate_chain_for_server_name* being invoked a second time, or *Connection.data_from_wire* being invoked after a connection has been invalidated due to incorrect data.

class InvalidAlertLevel
Raised when an alert code is not allowed to have the specified alert level.

class LeafCertificateHasNoPrivateKeyError
Raised when the leaf certificate doesn't have a private key.

class MoreThanOneLeafCertificateError
Raised when there's more than one leaf certificate in a set of certificates.

class NoLeafCertificateError
Raised when there are no leaf certificates in a set of certificates. A "leaf" is defined as a non-CA certificate.

class NoCertificateChainError
A certificate chain cannot be found between a specified leaf and a specified root.

class ExtraneousPrivateKeyError
A private key was found associated with a certificate when it shouldn't have been.

CHAPTER 4

TODO

- Certificates
  - TODO: design factories for building sets or chains of certificates from PEM files that are strict about:
    - private keys where they don't belong, or lack of private keys where we should have them
    - chain files that have things that aren't a part of the chain
- Determine better names for methods
- look through the past ten years of CVEs on OpenSSL, SecureTransport, GnuTLS, PolarSSL, etc.
  - old TLS Finished security flaw, having to do with half-closed sockets.
  - timing attacks: http://armoredbarista.blogspot.de/2014/04/easter-hack-even-more-critical-bugs-in.html
- Determine if the TLS implementation needs a clock (are there specific timeouts we need to wait for, etc).
  - look up what the requirements for responding to a handshake. scenario: client sends ClientHello (to renegotiate), server already had a huge amount of data in its write buffer. how long should client wait to receive ServerHello?
- add a method to ServerCertificates for non-SNI case
  - Actually, I don't think we need one yet. What's the use case for dynamic lookup of *non*-SNI server certificates?
- alerts
  - perhaps alert() should be removed.
  - figure out which TLS Alerts actually matter.
  - make alert take constants for level and code instead of integers.
- pin against port and host (???)
- connections should probably have a .cipher_suite, .tls_version, .session_id, .tls_extensions, and lots more
- allow disabling certain options (tls versions or algorithm choices) that we know are less secure than mandatory options.
- alternative cert validation support such as DANE or TACK.

CHAPTER 5

Future Work

- Session resumption:
  - ensure there's a solid way to invalidate session-resumption data on receipt of an alert (on both client and server)
- maybe allow clients to request renegotiation, if there are good use cases.
- maybe allow servers to request renegotiation, if there are good use cases.
- Is there a use case for making dh_params per-server-cert-chain in the SNI case? Some rumblings in this area, but no clear reason.

Table 5.1: A TLS full handshake

No.	Client	Server
1	Send: - ClientHello	<wait for ClientHello>
2	<wait for ServerHelloDone>	Send: • ServerHello • Certificate* • ServerKeyExchange* • CertificateRequest* • ServerHelloDone
3	Send: • Certificate* • ClientKeyExchange • CertificateVerify* • [ChangeCipherSpec] • Finished	<wait for client's Finished>
4	<wait for server's Finished>	Send: • [ChangeCipherSpec] • Finished

CHAPTER 6

Hello Messages

- ClientHello and ServerHello establish:
  - Protocol version
  - Session ID
  - Cipher suite
  - Compression method
- And generate:
  - ClientHello.random
  - ServerHello.random
- When the client sends a ClientHello to server, it can either respond with a ServerHello, or ignore it, leading to a fatal error and the closing of the connection.

2.Server:

- Certificate:
  - If it is to be authenticated
- ServerKeyExchange:
  - If server doesn't have a certificate, or
  - if server's certificate is for signing only
- CertificateRequest:
  - If:
    - server is authenticated, and
    - it is appropriate to the cipher suite selected
- ServerHelloDone:
  - To indicate that the hello-message phase of the handshake is complete

3.Client:

- Certificate:
  - If the server sent a CertificateRequest
- ClientKeyExchange:
  - The content of this message depends on the public key algorithm selected between ClientHello and ServerHello
- CertificateVerify:
  - If the client certificate sent is with signing ability
- Digitally signed
  - Verifies the possession of private key in certificate
- ChangeCipherSpec:
  - Send this and copy the pending cipher spec into the current cipher spec
- Finished
  - Sent under the new algorithms, keys, and secrets

4.Server:

- ChangeCipherSpec:
  - Send this and copy the pending cipher spec into the current cipher spec
- Finished
  - Sent under the new cipher spec

Session Resumption:

No.	Client	Server
1	Send: • ClientHello	<wait for ClientHello>
2	<wait for Server's Finished>	Check session cache for a match. • If the session ID is not found: – Generate a new session ID and perform a full handshake • If the session ID is found: – Is willing to re-establish the connection under the specified session state: ∗ If No: · Generate a new session ID and perform a full handshake ∗ If Yes, proceed to 3.
3	<wait for server's Finished>	Send: • ServerHello • [ChangeCipherSpec] • Finished
4	Send: • [ChangeCipherSpec] • Finished	<wait for client's Finished>

1.Client:

- ClientHello:
  - Sent using the session ID of the session to be resumed.

3.Server:

- ServerHello:
  - Sent with the same Session ID value (as in the ClientHello message).

Server as a state machine:

Input	Current State	Next State	Output
ClientHello	IDLE	CHECK_SESSION_CACHE	–
`id_found_somehow`	CHECK_SESSION_CACHE	WAIT_RESUME	(ServerHello, [ChangeCipherSpec], Finished)
`id_not_found_somehow`	CHECK_SESSION_CACHE	WAIT	(ServerHello, Certificate*, ServerKeyExchange*, CertificateRequest*, ServerHelloDone)
Finished (from Client)	WAIT	APP_DATA	([ChangeCipherSpec], Finished)
Finished (from Client)	WAIT_RESUME	APP_DATA	–
ClientHello	APP_DATA	APP_DATA	Alert(no_renegotiation)

Client as a state machine:

Input	Current State	Next State	Output
–	IDLE	WAIT_1	ClientHello
ServerHelloDone	WAIT_1	WAIT_2	(Certificate*, ClientKeyExchange, CertificateVerify*, [ChangeCipherSpec], Finished)
Finished (from Server)	WAIT_1	APP_DATA	([ChangeCipherSpec], Finished)
Finished (from Server)	WAIT_2	APP_DATA	–
HelloRequest	APP_DATA	APP_DATA	Alert(no_renegotiation)

* Indicates optional or situation-dependent messages that are not always sent.

Note: To help avoid pipeline stalls, ChangeCipherSpec is an independent TLS protocol content type, and is not actually a TLS handshake message.

Common states for both state machines:

Input	Current State	Next State	Output
Alert(close_notify)	APP_DATA	SHUTDOWN	(Alert(close_notify), `close_callback(False)`, `indicate_EOF_to_the_application_somehow`)
`Session.alert(close_notify)`	APP_DATA	HOST_INITIATED_CLOSING	Alert(close_notify)
Alert(close_notify)	HOST_INITIATED_CLOSING	SHUTDOWN	(`close_callback(True)`, `indicate_EOF_to_the_application_somehow`)
`Session.write_data`	APP_DATA	APP_DATA	`write_callback()`

CHAPTER 16

tls

16.1 tls package

16.1.1 Subpackages

tls.test package

Submodules

tls.test.test_alert module

tls.test.test_ciphersuites module

tls.test.test_hello_message module

class tls.test.test_hello_message.**TestClientHello**
Bases: object

Tests for the parsing of ClientHello messages.

test_as_bytes_client_hello_cipher_suites()
tls.hello_message.ClientHello() fails to construct a packet whose cipher_suites would be too short.

test_as_bytes_client_hello_compression_methosds_too_short()
tls.hello_message.ClientHello() fails to construct a packet whose compression_methods would be too short.

test_as_bytes_no_extensions()
ClientHello.as_bytes() returns the bytes it was created with

test_as_bytes_with_extensions()
ClientHello.as_bytes() returns the bytes it was created with

test_parse_client_hello_cipher_suites()
tls.hello_message.ClientHello() fails to parse a packet whose cipher_suites is too short.

test_parse_client_hello_compression_methods_too_short()
tls.hello_message.ClientHello() fails to parse a packet whose compression_methods is too short.

test_resumption_no_extensions()
parse_client_hello() returns an instance of ClientHello.

class tls.test.test_hello_message.**TestServerHello**
Bases: object

Tests for the parsing of ServerHello messages.

test_as_bytes_no_extensions()
ServerHello.as_bytes() returns the bytes it was created with

test_as_bytes_with_extensions()
ServerHello.as_bytes() returns the bytes it was created with

test_parse_server_hello()
parse_server_hello() returns an instance of ServerHello.

test_parse_server_hello_extensions()
parse_server_hello() returns an instance of ServerHello.

tls.test.test_message module

class tls.test.test_message.**TestASN1CertificateSerialization**
Bases: object

Tests for serializing *tls.message.ASN1Cert*

test_as_bytes()
tls.message.ASN1Cert.as_bytes() constructs a valid packet.

test_as_bytes_too_long()
tls.message.ASN1Cert.as_bytes() fails to construct a packet whose asn1_cert would be too long.

test_as_bytes_too_short()
tls.message.ASN1Cert.as_bytes() fails to construct a packet whose asn1_cert would be too short.

class tls.test.test_message.**TestCertificateParsing**
Bases: object

Tests for parsing of *tls.message.Certificate* messages.

test_as_bytes()
tls.message.Certificate.as_bytes() returns a valid packet.

test_as_bytes_too_long()
tls.message.Certificate.as_bytes() fails to construct a packet whose certificate_list would be too short.

test_as_bytes_too_short()
tls.message.Certificate.as_bytes() fails to construct a packet whose certificate_list would be too long.

test_parse_certificate()
tls.message.Certificate.from_bytes() parses a valid packet.

test_parse_certificate_too_long()
tls.message.Certificate.from_bytes() rejects a packet whose certificate_list is too long.

test_parse_certificate_too_short()
tls.message.Certificate.from_bytes() rejects a packet whose certificate_list is too short.

class tls.test.test_message.**TestCertificateRequestParsing**
Bases: object

Tests for parsing of CertificateRequest messages.

test_as_bytes_certificate_types_too_short()
tls.message.CertificateRequest() fails to construct a certificate request packet whose certificate_types would be too short.

test_as_bytes_supported_signature_algorithms_too_short()
CertificateRequest() fails to construct a certificate request packet whose supported_signature_algorithms would be too short.

test_parse_certificate_types_too_short()
tls.message.CertificateRequest() fails to parse a certificate request packet whose certificate_types is too short.

test_parse_supported_signature_algorithms_too_short()
CertificateRequest() fails to parse a certificate request packet whose supported_signature_algorithms is too short.

class tls.test.test_message.**TestHandshakeStructParsing**
Bases: object

Tests for parsing of tls.messages.Handshake structs.

class tls.test.test_message.**TestPreMasterSecretParsing**
Bases: object

Tests for parsing of PreMasterSecret struct.

class tls.test.test_message.**TestServerDHParamsparsing**
Bases: object

Tests for parsing of ServerDHParams struct.

tls.test.test_record module

class tls.test.test_record.**TestTLSCiphertextParser**
Bases: object

Tests for parsing of TLSCiphertext records.

test_fragment_too_long()
TLSCiphertext() rejects a packet containing a longer-than-allowed fragment.

test_parse_tls_ciphertext_handshake()
TLSCiphertext, which has attributes representing all the fields in the TLSCiphertext struct.

class tls.test.test_record.**TestTLSCompressedParsing**
Bases: object

Tests for parsing of TLSCompressed records.

test_fragment_too_long()
tls.record.TLSCompressed() rejects a packet containing a longer-than-allowed fragment.

test_incomplete_packet()
Reject an incomplete packet

test_not_enough_data_to_fragment()
Detect insufficient data to fragment.

test_parse_tls_compressed_handshake()
`TLSCompressed`, which has attributes representing all the fields in the TLSCompressed struct.

test_parse_tls_compressed_wrong_type()
Raise an error when the type is not one of those defined in ContentType

class tls.test.test_record.**TestTLSPlaintextParsing**
Bases: `object`

Tests for parsing of TLSPlaintext records.

test_as_bytes()
Construct a TLSPlaintext object as bytes.

test_as_bytes_fragment_too_long()
tls.record.TLSPlaintext() fails to construct a packet with a longer-than-allowed fragment.

test_incomplete_packet()
Reject an incomplete packet

test_not_enough_data_to_fragment()
Detect insufficient data to fragment.

test_parse_fragment_too_long()
tls.record.TLSPlaintext() fails to parse a packet containing a longer-than-allowed fragment.

test_parse_tls_plaintext_handshake()
`parse_tls_plaintext()` returns an instance of `TLSPlaintext`, which has attributes representing all the fields in the TLSPlaintext struct.

test_parse_tls_plaintext_wrong_type()
Raise an error when the type is not one of those defined in ContentType

tls.test.test_utils module

class tls.test.test_utils.**Equals5**(*subcon*)
Bases: `construct.adapters.Validator`

A test fixture `construct.adapters.Validator` subclass that ensures a numeric field equals 5.

class tls.test.test_utils.**IntegerEnum**
Bases: `enum.Enum`

An enum of `int` instances. Used as a test fixture.

class tls.test.test_utils.**TestBytesAdapter**
Bases: `object`

Tests for `tls.utils.BytesAdapter`.

bytes_adapted()
A `tls.utils.BytesAdapter` that adapts a trivial `construct.Construct()`.

test_decode_passes_value_through(*bytes_adapted*, *value*)
`tls.utils.BytesAdapter._decode()` decodes `bytes` as `bytes`.

test_encode_allows_bytes(*bytes_adapted*, *byte_string*)
tls.utils.BytesAdapter._encode() encodes bytes without raising an exception.

test_encode_disallows_non_bytes(*bytes_adapted*, *non_bytes*)
tls.utils.BytesAdapter._encode() raises a construct.core.AdaptationError when encoding anything that isn't bytes.

class tls.test.test_utils.**TestEnumClass**
Bases: object

Tests for *tls.utils.EnumClass()*.

UBInt8Enum()
A *tls.utils.EnumClass()* that adapts *IntegerEnum*'s members to UBInt8().

test_build(*UBInt8Enum*)
tls.utils.EnumClass() encodes members of its enum according to its construct.

test_build_enum_has_wrong_type(*UBInt8Enum*)
tls.utils.EnumClass() raises construct.adapters.MappingError when encoding something that isn't a member of its enum.

test_parse(*UBInt8Enum*)
tls.utils.EnumClass() decodes a binary sequence as members of its enum via its construct.

class tls.test.test_utils.**TestEnumSwitch**
Bases: object

Tests for *tls.utils.EnumSwitch()*.

UBInt8EnumMappedStruct()
A construct.core.Struct containing an *tls.utils.EnumSwitch()* that switches on *IntegerEnum*. The struct's value field varies depending on the value of its type and the corresponding enum member specified in the value_choices dictionary passed to the *tls.utils.EnumSwitch()*.

test_build(*UBInt8EnumMappedStruct*, *type_*, *value*, *encoded*)
A struct that contains *tls.utils.EnumSwitch()* encodes its value_field according to the enum member specified in its type_field.

test_parse(*UBInt8EnumMappedStruct*, *type_*, *value*, *encoded*)
A struct that contains *tls.utils.EnumSwitch()* decodes its value field according to the enum member specified by its type_field.

test_round_trip(*UBInt8EnumMappedStruct*, *type_*, *value*, *encoded*)
A struct that contains *tls.utils.EnumSwitch()* decodes a binary sequence encoded by a struct with that same *tls.utils.EnumSwitch()* and vice versa.

class tls.test.test_utils.**TestPrefixedBytesWithDefaultLength**
Bases: object

Tests for *tls.utils.PrefixedBytes()* with the default construct.macros.UBInt8() length_field construct.

prefixed_bytes()
A trivial *tls.utils.PrefixedBytes()* construct with the default construct.macros.UBInt8() length field.

test_build(*prefixed_bytes*, *bytestring*, *encoded*)
tls.utils.PrefixedBytes() encodes bytes as a length-prefixed byte sequence.

test_parse(*prefixed_bytes*, *bytestring*, *encoded*)
tls.utils.PrefixedBytes() decodes a length-prefixed byte sequence as bytes.

test_round_trip(*prefixed_bytes*, *bytestring*, *encoded*)
tls.utils.PrefixedBytes() decodes a length-prefixed binary sequence encoded by *tls.utils.PrefixedBytes()* and vice versa.

class tls.test.test_utils.**TestPrefixedBytesWithOverriddenLength**
Bases: object

Tests for *tls.utils.PrefixedBytes()* with a user-supplied length_field construct.

test_build(*bytestring*, *encoded*, *length_field*)
tls.utils.PrefixedBytes() uses the supplied length_field to encode bytes as a length-prefix binary sequence.

test_parse(*bytestring*, *encoded*, *length_field*)
tls.utils.PrefixedBytes() decodes a length-prefixed binary sequence into bytes according to the supplied length_field.

test_round_trip(*bytestring*, *encoded*, *length_field*)
tls.utils.PrefixedBytes() decodes a length-prefixed binary sequence encoded by *tls.utils.PrefixedBytes()* when the two share a length_field and vice versa.

class tls.test.test_utils.**TestTLSPrefixedArray**
Bases: object

Tests for *tls.utils.TLSPrefixedArray()*.

test_build(*tls_array*, *ints*, *uint8_encoded*)
A *tls.utils.TLSPrefixedArray()* specialized on a given construct.Construct() encodes a sequence of objects as a 16-bit length followed by each object as encoded by that construct.

test_parse(*tls_array*, *ints*, *uint8_encoded*)
A *tls.utils.TLSPrefixedArray()* specialized on a given construct.Construct() decodes a binary sequence, prefixed by its 16-bit length, as a list of objects decoded by that construct.

test_round_trip(*tls_array*, *ints*, *uint8_encoded*)
A *tls.utils.TLSPrefixedArray()* decodes a length-prefixed binary sequence encoded by a *tls.utils.TLSPrefixedArray()* specialized on the same construct and vice versa.

tls_array()
A *tls.utils.TLSPrefixedArray()* of construct.macros.UBInt8().

class tls.test.test_utils.**TestTLSPrefixedArrayWithLengthValidator**
Bases: object

Tests for *tls.utils.TLSPrefixedArray* with a length_validator.

TLSUBInt8Array()
A *tls.utils.TLSPrefixedArray* specialized on construct.macros.UBInt8()

TLSUBInt8Length5Array()
Like TLSPrefixedArrayWithLengthValidator.TLSUBInt8Length5Array(), but only accepts arrays of length 5.

test_build_invalid(*TLSUBInt8Length5Array*, *invalid*)
tls.utils.TLSPrefixedArray raises a construct.adapters.ValidationError when encoding a list with an invalid length.

test_build_valid(*TLSUBInt8Length5Array*, *TLSUBInt8Array*)
tls.utils.TLSPrefixedArray encodes an array that passes validation.

test_parse_invalid(*TLSUBInt8Length5Array*, *invalid*)
tls.utils.TLSPrefixedArray raises a construct.adapters.ValidationError when decoding an array with an invalid length.

test_parse_valid(*TLSUBInt8Length5Array*, *TLSUBInt8Array*)
tls.utils.TLSPrefixedArray decodes an array that passes validation.

class tls.test.test_utils.**UnicodeEnum**
Bases: enum.Enum

An enum of str (or unicode) instances. Used as a test fixture.

tls.test.test_utils.**test_size_at_least_validate**(*min_size*, *num*, *acceptable*)
SizeAtLeast._validate() enforces its minimum size inclusively when encoding numbers.

tls.test.test_utils.**test_size_at_most_validate**(*max_size*, *num*, *acceptable*)
SizeAtMost._validate() enforces its maximum size inclusively when encoding numbers.

tls.test.test_utils.**test_size_within_validate**(*min_size*, *max_size*, *num*, *acceptable*)
SizeWithin._validate() enforces its maximum size inclusively when encoding numbers.

Module contents

16.1.2 Submodules

16.1.3 tls.alert_message module

class tls.alert_message.**Alert**(*level*, *description*)
Bases: object

An object representing an Alert message.

classmethod from_bytes(*bytes*)
Parse an Alert struct.

Parameters **bytes** – the bytes representing the input.

Returns Alert object.

16.1.4 tls.ciphersuites module

16.1.5 tls.exceptions module

16.1.6 tls.hello_message module

class tls.hello_message.**ClientHello**(*client_version*, *random*, *session_id*, *cipher_suites*, *compression_methods*, *extensions*)
Bases: object

An object representing a ClientHello message.

classmethod from_bytes(*bytes*)
Parse a ClientHello struct.

Parameters **bytes** – the bytes representing the input.

Returns ClientHello object.

class tls.hello_message.**Extension**(*type*, *data*)
Bases: object

An object representing an Extension struct.

class tls.hello_message.**ProtocolVersion**(*major*, *minor*)
Bases: object

An object representing a ProtocolVersion struct.

class tls.hello_message.**Random**(*gmt_unix_time*, *random_bytes*)
Bases: object

An object representing a Random struct.

class tls.hello_message.**ServerHello**(*server_version*, *random*, *session_id*, *cipher_suite*, *compression_method*, *extensions*)
Bases: object

An object representing a ServerHello message.

classmethod from_bytes(*bytes*)
Parse a ServerHello struct.

Parameters **bytes** – the bytes representing the input.

Returns ServerHello object.

16.1.7 tls.message module

class tls.message.**ASN1Cert**(*asn1_cert*)
Bases: object

An object representing ASN.1 Certificate

class tls.message.**Certificate**(*certificate_list*)
Bases: object

An object representing a Certificate struct.

classmethod from_bytes(*bytes*)
Parse a Certificate struct.

Parameters **bytes** – the bytes representing the input.

Returns Certificate object.

class tls.message.**CertificateRequest**(*certificate_types*, *supported_signature_algorithms*, *certificate_authorities*)
Bases: object

An object representing a CertificateRequest struct.

classmethod from_bytes(*bytes*)
Parse a CertificateRequest struct.

Parameters **bytes** – the bytes representing the input.

Returns CertificateRequest object.

class tls.message.**Handshake**(*msg_type*, *length*, *body*)
Bases: object

An object representing a Handshake struct.

classmethod from_bytes(*bytes*)
Parse a Handshake struct.

Parameters **bytes** – the bytes representing the input.

Returns Handshake object.

class tls.message.**HelloRequest**
Bases: object

An object representing a HelloRequest struct.

class tls.message.**PreMasterSecret**(*client_version*, *random*)
Bases: object

An object representing a PreMasterSecret struct.

classmethod from_bytes(*bytes*)
Parse a PreMasterSecret struct.

Parameters **bytes** – the bytes representing the input.

Returns CertificateRequest object.

class tls.message.**ServerDHParams**(*dh_p*, *dh_g*, *dh_Ys*)
Bases: object

An object representing a ServerDHParams struct.

classmethod from_bytes(*bytes*)
Parse a ServerDHParams struct.

Parameters **bytes** – the bytes representing the input.

Returns ServerDHParams object.

class tls.message.**ServerHelloDone**
Bases: object

An object representing a ServerHelloDone struct.

class tls.message.**SignatureAndHashAlgorithm**(*hash*, *signature*)
Bases: object

An object representing a SignatureAndHashAlgorithm struct.

16.1.8 tls.record module

class tls.record.**ProtocolVersion**(*major*, *minor*)
Bases: object

An object representing a ProtocolVersion struct.

class tls.record.**TLSCiphertext**(*type*, *version*, *fragment*)
Bases: object

An object representing a TLSCiphertext struct.

classmethod from_bytes(*bytes*)
Parse a TLSCiphertext struct.

Parameters **bytes** – the bytes representing the input.

Returns TLSCiphertext object.

class tls.record.**TLSCompressed**(*type*, *version*, *fragment*)
Bases: object

An object representing a TLSCompressed struct.

classmethod from_bytes(*bytes*)
Parse a TLSCompressed struct.

Parameters **bytes** – the bytes representing the input.

Returns TLSCompressed object.

class tls.record.**TLSPlaintext**(*type*, *version*, *fragment*)
Bases: object

An object representing a TLSPlaintext struct.

classmethod from_bytes(*bytes*)
Parse a TLSPlaintext struct.

Parameters **bytes** – the bytes representing the input.

Returns TLSPlaintext object.

16.1.9 tls.utils module

tls.utils.**EnumClass**(*type_field*, *type_enum*)
Maps the members of an enum.Enum to a single kind of construct.Construct.

Parameters

- **type_field** (construct.Construct) – The construct that represents the enum's members. The type of this should correspond to the enum members' types; for instance, an enum with a maximum value of 65535 would use a construct.macros.UBInt16.
- **type_enum** (enum.Enum) – The enum to encode and decode.

tls.utils.**EnumSwitch**(*type_field*, *type_enum*, *value_field*, *value_choices*)
Maps the members of an enum.Enum to arbitrary construct.Constructs(). It returns a tuple intended to be spliced into another construct.Construct()'s definition:

```
>>> from tls.utils import EnumSwitch
>>> import construct, enum
>>> class IntEnum(enum.Enum):
...     VALUE = 1
...
>>> construct.Struct(
...     "name",
...     construct.UBInt8("an_integer"),
...     *EnumSwitch(type_field=construct.UBInt8("type"),
...                 type_enum=IntEnum,
...                 value_field="value",
...                 value_choices={
...                     IntEnum.VALUE: construct.UBInt8("first"),
...      })
... )
...
Struct('name')
```

Parameters

- **type_field** (construct.Construct) – The construct that represents the enum's members. The type of this should correspond to the enum members' types, so an enum with a maximum value of 65535, for example, would use a construct.macros.UBInt16.
- **type_enum** (enum.Enum) – The enum to encode and decode.
- **value_field** (str) – The attribute name under which this value will be accessible.

- **value_choices** (dict) – A dictionary that maps members of *type_enum* to subconstructs. This follows construct.core.Switch()'s API, so _default_ will match any members without an explicit mapping.

Returns A tuple of the form (*EnumClass()*, construct.core.Switch())

tls.utils.**PrefixedBytes**(*name*, *length_field=FormatField('length')*)

Length-prefixed binary data. This is like a construct.macros.PascalString() that raises a constrcut.AdaptationError when encoding something other than bytes.

Parameters

- **name** (str) – The attribute name under which this value will be accessible.
- **length_field** (a construct.core.FormatField) – (optional) The prefixed length field. Defaults to construct.macros.UBInt8().

class tls.utils.**SizeAtLeast**(*subconn*, *min_size*)

Bases: construct.adapters.Validator

A construct.adapter.Validator that validates a sequence size is greater than or equal to some minimum.

```
>>> from construct import UBInt8
>>> from tls.utils import SizeAtLeast, PrefixedBytes
>>> PrefixedBytes(None, SizeAtLeast(UBInt8("length"),
...                     min_size=2)).parse(b'{a')
Traceback (most recent call last):
    ...
construct.core.ValidationError: ('invalid object', b'a')
```

Parameters

- **subcon** (construct.core.Construct) – the construct to validate.
- **min_size** (int) – the (inclusive) minimum allowable size for the validated sequence.

class tls.utils.**SizeAtMost**(*subconn*, *max_size*)

Bases: construct.adapters.Validator

A construct.adapter.Validator that validates a sequence size is less than or equal to some maximum.

```
>>> from tls.utils import SizeAtMost, PrefixedBytes
>>> PrefixedBytes(None, SizeAtMost(UBInt8("length"),
...                     max_size=1)).parse(b'}aa')
Traceback (most recent call last):
    ...
construct.core.ValidationError: ('invalid object', b'}aa')
```

Parameters

- **subcon** (construct.core.Construct) – the construct to validate.
- **max_size** (int) – the (inclusive) maximum allowable size for the validated sequence.

class tls.utils.**SizeWithin**(*subconn*, *min_size*, *max_size*)

Bases: construct.adapters.Validator

A construct.adapter.Validator that validates a sequence's size is within some bounds. The bounds are inclusive.

```
>>> from tls.utils import SizeWithin, PrefixedBytes
>>> PrefixedBytes(None, SizeWithin(UBInt8("length"),
...                     min_size=2, max_size=2)).parse(b'{a')
Traceback (most recent call last):
    ...
construct.core.ValidationError: ('invalid object', b'{a')
```

Parameters

- **subcon** (construct.core.Construct) – the construct to validate.
- **min_size** (int) – the (inclusive) minimum allowable size for the validated sequence.
- **max_size** (int) – the (inclusive) maximum allowable size for the validated sequence.

tls.utils.**TLSPrefixedArray**(*subconn*, *length_name='length'*, *length_validator=None*)
The TLS vector type. It specializes on another construct.Construct and then encodes or decodes an arbitrarily long list or array of those constructs, prepending or reading a leading 16 bit length.

Parameters

- **subconn** (construct.Construct) – The construct this array contains.
- **length_name** (str) – (optional) The attribute name under which the construct.macros.UBInt16 representing this array's length will be accessible. You do not need to provide this when encoding a python sequence!
- **length_validator** () – (optional) A callable that validates the array's length construct.

tls.utils.**UBInt24**(*name*)
A 24-bit integer.

Parameters **name** (str) – The attribute name under which this value will be accessible.

16.1.10 Module contents

t

R

S

T

U