tlslite-ng Documentation

Release 0.8.0-alpha38

Hubert Kario

Oct 22, 2020

Contents

Contents:

CHAPTER 1

tlslite

1.1 tlslite package

TLS Lite is a free python library that implements SSL and TLS. TLS Lite supports RSA and SRP ciphersuites. TLS Lite is pure python, however it can use other libraries for faster crypto operations. TLS Lite integrates with several stdlib neworking libraries.

API documentation is available in the 'docs' directory.

If you have questions or feedback, feel free to contact me.

To use, do:

```
from tlslite import TLSConnection, ...
```

If you want to import the most useful objects, the cleanest way is:

```
from tlslite.api import *
```

Then use the `TLSConnection` class with a socket. (Or, use one of the integration classes in *tlslite.integration*).

1.1.1 Subpackages

tlslite.integration package

Classes for integrating TLS Lite with other packages.

Submodules

tlslite.integration.asyncstatemachine module

A state machine for using TLS Lite with asynchronous I/O.

class tlslite.integration.asyncstatemachine.**AsyncStateMachine**
Bases: object

This is an abstract class that's used to integrate TLS Lite with asyncore and Twisted.

This class signals wantsReadsEvent() and wantsWriteEvent(). When the underlying socket has become readable or writeable, the event should be passed to this class by calling inReadEvent() or inWriteEvent(). This class will then try to read or write through the socket, and will update its state appropriately.

This class will forward higher-level events to its subclass. For example, when a complete TLS record has been received, outReadEvent() will be called with the decrypted data.

__init__()
Initialize self. See help(type(self)) for accurate signature.

inReadEvent()
Tell the state machine it can read from the socket.

inWriteEvent()
Tell the state machine it can write to the socket.

outCloseEvent()
Called when a close operation completes.

May be overridden in subclass.

outConnectEvent()
Called when a handshake operation completes.

May be overridden in subclass.

outReadEvent(*readBuffer*)
Called when a read operation completes.

May be overridden in subclass.

outWriteEvent()
Called when a write operation completes.

May be overridden in subclass.

setCloseOp()
Start a close operation.

setHandshakeOp(*handshaker*)
Start a handshake operation.

Parameters **handshaker** (*generator*) – A generator created by using one of the asynchronous handshake functions (i.e. *handshakeServerAsync()* , or handshakeClientxxx(..., async_=True).

setServerHandshakeOp(**args*)
Start a handshake operation.

The arguments passed to this function will be forwarded to *handshakeServerAsync*.

setWriteOp(*writeBuffer*)
Start a write operation.

Parameters **writeBuffer** (*str*) – The string to transmit.

wantsReadEvent()
If the state machine wants to read.

If an operation is active, this returns whether or not the operation wants to read from the socket. If an operation is not active, this returns None.

Return type bool or None

Returns If the state machine wants to read.

wantsWriteEvent()
If the state machine wants to write.

If an operation is active, this returns whether or not the operation wants to write to the socket. If an operation is not active, this returns None.

Return type bool or None

Returns If the state machine wants to write.

tlslite.integration.clienthelper module

A helper class for using TLS Lite with stdlib clients (httplib, xmlrpclib, imaplib, poplib).

class tlslite.integration.clienthelper.**ClientHelper**(*username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*, *anon=False*, *host=None*)

Bases: object

This is a helper class used to integrate TLS Lite with various TLS clients (e.g. poplib, smtplib, httplib, etc.)

__init__(*username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*, *anon=False*, *host=None*)
For client authentication, use one of these argument combinations:

- username, password (SRP)
- certChain, privateKey (certificate)

For server authentication, you can either rely on the implicit mutual authentication performed by SRP, or you can do certificate-based server authentication with one of these argument combinations:

- x509Fingerprint

Certificate-based server authentication is compatible with SRP or certificate-based client authentication.

The constructor does not perform the TLS handshake itself, but simply stores these arguments for later. The handshake is performed only when this class needs to connect with the server. Then you should be prepared to handle TLS-specific exceptions. See the client handshake functions in *TLSConnection* for details on which exceptions might be raised.

Parameters

- **username** (*str*) – SRP username. Requires the 'password' argument.
- **password** (*str*) – SRP password for mutual authentication. Requires the 'username' argument.
- **certChain** (X509CertChain) – Certificate chain for client authentication. Requires the 'privateKey' argument. Excludes the SRP arguments.

- **privateKey** (RSAKey) – Private key for client authentication. Requires the 'certChain' argument. Excludes the SRP arguments.
- **checker** (Checker) – Callable object called after handshaking to evaluate the connection and raise an Exception if necessary.
- **settings** (HandshakeSettings) – Various settings which can be used to control the ciphersuites, certificate types, and SSL/TLS versions offered by the client.
- **anon** (*bool*) – set to True if the negotiation should advertise only anonymous TLS ciphersuites. Mutually exclusive with client certificate authentication or SRP authentication
- **host** (*str or None*) – the hostname that the connection is made to. Can be an IP address (in which case the SNI extension won't be sent). Can include the port (in which case the port will be stripped and ignored).

tlslite.integration.httptlsconnection module

TLS Lite + httplib.

class tlslite.integration.httptlsconnection.**HTTPTLSConnection**(*host*, *port=None*, *strict=None*, *timeout=<object object>*, *source_address=None*, *username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*, *ignoreAbruptClose=False*, *anon=False*)

Bases: http.client.HTTPConnection, *tlslite.integration.clienthelper.ClientHelper*

This class extends L{httplib.HTTPConnection} to support TLS.

__init__(*host*, *port=None*, *strict=None*, *timeout=<object object>*, *source_address=None*, *username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*, *ignoreAbruptClose=False*, *anon=False*)

Create a new HTTPTLSConnection.

For client authentication, use one of these argument combinations:

- username, password (SRP)
- certChain, privateKey (certificate)

For server authentication, you can either rely on the implicit mutual authentication performed by SRP or you can do certificate-based server authentication with one of these argument combinations:

- x509Fingerprint

Certificate-based server authentication is compatible with SRP or certificate-based client authentication.

The constructor does not perform the TLS handshake itself, but simply stores these arguments for later. The handshake is performed only when this class needs to connect with the server. Thus you should be prepared

to handle TLS-specific exceptions when calling methods inherited from `httplib.HTTPConnection` such as request(), connect(), and send(). See the client handshake functions in *TLSConnection* for details on which exceptions might be raised.

Parameters

- **host** (*str*) – Server to connect to.
- **port** (*int*) – Port to connect to.
- **username** (*str*) – SRP username. Requires the 'password' argument.
- **password** (*str*) – SRP password for mutual authentication. Requires the 'username' argument.
- **certChain** (X509CertChain) – Certificate chain for client authentication. Requires the 'privateKey' argument. Excludes the SRP arguments.
- **privateKey** (RSAKey) – Private key for client authentication. Requires the 'certChain' argument. Excludes the SRP arguments.
- **checker** (Checker) – Callable object called after handshaking to evaluate the connection and raise an Exception if necessary.
- **settings** (HandshakeSettings) – Various settings which can be used to control the ciphersuites, certificate types, and SSL/TLS versions offered by the client.
- **ignoreAbruptClose** (*bool*) – ignore the TLSAbruptCloseError on unexpected hangup.

__module__ = 'tlslite.integration.httptlsconnection'

connect()
Connect to the host and port specified in __init__.

tlslite.integration.imap4_tls module

TLS Lite + imaplib.

class tlslite.integration.imap4_tls.**IMAP4_TLS**(*host=''*, *port=993*, *username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*)
Bases: imaplib.IMAP4, *tlslite.integration.clienthelper.ClientHelper*

This class extends imaplib.IMAP4 with TLS support.

__init__(*host=''*, *port=993*, *username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*)
Create a new IMAP4_TLS.

For client authentication, use one of these argument combinations:

- username, password (SRP)
- certChain, privateKey (certificate)

For server authentication, you can either rely on the implicit mutual authentication performed by SRP or you can do certificate-based server authentication with one of these argument combinations:

- x509Fingerprint

Certificate-based server authentication is compatible with SRP or certificate-based client authentication.

The caller should be prepared to handle TLS-specific exceptions. See the client handshake functions in *TLSConnection* for details on which exceptions might be raised.

Parameters

- **host** (*str*) – Server to connect to.
- **port** (*int*) – Port to connect to.
- **username** (*str*) – SRP username. Requires the 'password' argument.
- **password** (*str*) – SRP password for mutual authentication. Requires the 'username' argument.
- **certChain** (X509CertChain) – Certificate chain for client authentication. Requires the 'privateKey' argument. Excludes the SRP arguments.
- **privateKey** (RSAKey) – Private key for client authentication. Requires the 'certChain' argument. Excludes the SRP arguments.
- **checker** (Checker) – Callable object called after handshaking to evaluate the connection and raise an Exception if necessary.
- **settings** (HandshakeSettings) – Various settings which can be used to control the ciphersuites, certificate types, and SSL/TLS versions offered by the client.

open(*host=''*, *port=993*)

Setup connection to remote server on "host:port".

This connection will be used by the routines: read, readline, send, shutdown.

tlslite.integration.pop3_tls module

TLS Lite + poplib.

class tlslite.integration.pop3_tls.**POP3_TLS**(*host*, *port=995*, *timeout=<object object>*, *username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*)

Bases: poplib.POP3, *tlslite.integration.clienthelper.ClientHelper*

This class extends poplib.POP3 with TLS support.

__init__(*host*, *port=995*, *timeout=<object object>*, *username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*)

Create a new POP3_TLS.

For client authentication, use one of these argument combinations:

- username, password (SRP)
- certChain, privateKey (certificate)

For server authentication, you can either rely on the implicit mutual authentication performed by SRP or you can do certificate-based server authentication with one of these argument combinations:

- x509Fingerprint

Certificate-based server authentication is compatible with SRP or certificate-based client authentication.

The caller should be prepared to handle TLS-specific exceptions. See the client handshake functions in *TLSConnection* for details on which exceptions might be raised.

Parameters

- **host** (*str*) – Server to connect to.
- **port** (*int*) – Port to connect to.
- **username** (*str*) – SRP username.
- **password** (*str*) – SRP password for mutual authentication. Requires the 'username' argument.
- **certChain** (X509CertChain) – Certificate chain for client authentication. Requires the 'privateKey' argument. Excludes the SRP argument.
- **privateKey** (RSAKey) – Private key for client authentication. Requires the 'certChain' argument. Excludes the SRP argument.
- **checker** (Checker) – Callable object called after handshaking to evaluate the connection and raise an Exception if necessary.
- **settings** (HandshakeSettings) – Various settings which can be used to control the ciphersuites, certificate types, and SSL/TLS versions offered by the client.

tlslite.integration.smtp_tls module

TLS Lite + smtplib.

class tlslite.integration.smtp_tls.**SMTP_TLS**(*host=''*, *port=0*, *local_hostname=None*, *timeout=<object object>*, *source_address=None*)

Bases: smtplib.SMTP

This class extends smtplib.SMTP with TLS support.

starttls(*username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*)

Puts the connection to the SMTP server into TLS mode.

If the server supports TLS, this will encrypt the rest of the SMTP session.

For client authentication, use one of these argument combinations:

- username, password (SRP)
- certChain, privateKey (certificate)

For server authentication, you can either rely on the implicit mutual authentication performed by SRP or you can do certificate-based server authentication with one of these argument combinations:

- x509Fingerprint

Certificate-based server authentication is compatible with SRP or certificate-based client authentication.

The caller should be prepared to handle TLS-specific exceptions. See the client handshake functions in *TLSConnection* for details on which exceptions might be raised.

Parameters

- **username** (*str*) – SRP username. Requires the 'password' argument.
- **password** (*str*) – SRP password for mutual authentication. Requires the 'username' argument.
- **certChain** (X509CertChain) – Certificate chain for client authentication. Requires the 'privateKey' argument. Excludes the SRP arguments.

- **privateKey** (RSAKey) – Private key for client authentication. Requires the 'certChain' argument. Excludes the SRP arguments.
- **checker** (Checker) – Callable object called after handshaking to evaluate the connection and raise an Exception if necessary.
- **settings** (HandshakeSettings) – Various settings which can be used to control the ciphersuites, certificate types, and SSL/TLS versions offered by the client.

tlslite.integration.tlsasyncdispatchermixin module

TLS Lite + asyncore.

class tlslite.integration.tlsasyncdispatchermixin.**TLSAsyncDispatcherMixIn**(*sock=None*)

Bases: *tlslite.integration.asyncstatemachine.AsyncStateMachine*

This class can be "mixed in" with an asyncore.dispatcher to add TLS support.

This class essentially sits between the dispatcher and the select loop, intercepting events and only calling the dispatcher when applicable.

In the case of *handle_read()*, a read operation will be activated, and when it completes, the bytes will be placed in a buffer where the dispatcher can retrieve them by calling *recv()*, and the dispatcher's *handle_read()* will be called.

In the case of *handle_write()*, the dispatcher's *handle_write()* will be called, and when it calls *send()*, a write operation will be activated.

To use this class, you must combine it with an asyncore.dispatcher, and pass in a handshake operation with setServerHandshakeOp().

Below is an example of using this class with medusa. This class is mixed in with http_channel to create http_tls_channel. Note:

1. the mix-in is listed first in the inheritance list
2. the input buffer size must be at least 16K, otherwise the dispatcher might not read all the bytes from the TLS layer, leaving some bytes in limbo.
3. IE seems to have a problem receiving a whole HTTP response in a single TLS record, so HTML pages containing 'rnrn' won't be displayed on IE.

Add the following text into 'start_medusa.py', in the 'HTTP Server' section:

```
from tlslite import *
s = open("./serverX509Cert.pem").read()
x509 = X509()
x509.parse(s)
cert_chain = X509CertChain([x509])

s = open("./serverX509Key.pem").read()
privateKey = parsePEMKey(s, private=True)

class http_tls_channel(TLSAsyncDispatcherMixIn,
                       http_server.http_channel):
    ac_in_buffer_size = 16384

    def __init__ (self, server, conn, addr):
        http_server.http_channel.__init__(self, server, conn, addr)
        TLSAsyncDispatcherMixIn.__init__(self, conn)
```

(continues on next page)

(continued from previous page)

```
        self.tlsConnection.ignoreAbruptClose = True
        self.setServerHandshakeOp(certChain=cert_chain,
                                  privateKey=privateKey)

hs.channel_class = http_tls_channel
```

If the TLS layer raises an exception, the exception will be caught in asyncore.dispatcher, which will call *close()* on this class. The TLS layer always closes the TLS connection before raising an exception, so the close operation will complete right away, causing asyncore.dispatcher.close() to be called, which closes the socket and removes this instance from the asyncore loop.

__init__(*sock=None*)
Initialize self. See help(type(self)) for accurate signature.

close()

handle_read()

handle_write()

outCloseEvent()
Called when a close operation completes.

May be overridden in subclass.

outConnectEvent()
Called when a handshake operation completes.

May be overridden in subclass.

outReadEvent(*readBuffer*)
Called when a read operation completes.

May be overridden in subclass.

outWriteEvent()
Called when a write operation completes.

May be overridden in subclass.

readable()

recv(*bufferSize=16384*)

send(*writeBuffer*)

writable()

tlslite.integration.tlssocketservermixin module

TLS Lite + SocketServer.

class tlslite.integration.tlssocketservermixin.**TLSSocketServerMixIn**
Bases: object

This class can be mixed in with any SocketServer.TCPServer to add TLS support.

To use this class, define a new class that inherits from it and some SocketServer.TCPServer (with the mix-in first). Then implement the *handshake()* method, doing some sort of server handshake on the connection argument. If the handshake method returns True, the RequestHandler will be triggered. Below is a complete example of a threaded HTTPS server:

```
from SocketServer import *
from BaseHTTPServer import *
from SimpleHTTPServer import *
from tlslite import *

s = open("./serverX509Cert.pem").read()
x509 = X509()
x509.parse(s)
cert_chain = X509CertChain([x509])

s = open("./serverX509Key.pem").read()
privateKey = parsePEMKey(s, private=True)

sessionCache = SessionCache()

class MyHTTPServer(ThreadingMixIn, TLSSocketServerMixIn,
                   HTTPServer):
  def handshake(self, tlsConnection):
      try:
          tlsConnection.handshakeServer(certChain=cert_chain,
                                        privateKey=privateKey,
                                        sessionCache=sessionCache)
          tlsConnection.ignoreAbruptClose = True
          return True
      except TLSError, error:
          print "Handshake failure:", str(error)
          return False

httpd = MyHTTPServer(('localhost', 443), SimpleHTTPRequestHandler)
httpd.serve_forever()
```

finish_request(*sock*, *client_address*)

handshake(*tlsConnection*)

tlslite.integration.xmlrpcserver module

xmlrpcserver.py - simple XML RPC server supporting TLS.

class tlslite.integration.xmlrpcserver.**MultiPathTLSXMLRPCServer**(*addr*, **args*, ***kwargs*)

Bases: *tlslite.integration.xmlrpcserver.TLSXMLRPCServer*

Multipath XML-RPC Server using TLS.

__init__(*addr*, **args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class tlslite.integration.xmlrpcserver.**TLSXMLRPCRequestHandler**(*request*, *client_address*, *server*)

Bases: xmlrpc.server.SimpleXMLRPCRequestHandler

XMLRPCRequestHandler using TLS.

do_POST()
Handle the HTTPS POST request.

setup()
Setup the connection for TLS.

class tlslite.integration.xmlrpcserver.**TLSXMLRPCServer**(*addr*, **args*, ***kwargs*)
Bases: *tlslite.integration.tlssocketservermixin.TLSSocketServerMixIn*, xmlrpc.server.SimpleXMLRPCServer

Simple XML-RPC server using TLS.

__init__(*addr*, **args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

tlslite.integration.xmlrpctransport module

TLS Lite + xmlrpclib.

class tlslite.integration.xmlrpctransport.**XMLRPCTransport**(*use_datetime=0*, *username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*, *ignoreAbruptClose=False*)
Bases: xmlrpc.client.Transport, *tlslite.integration.clienthelper.ClientHelper*

Handles an HTTPS transaction to an XML-RPC server.

__init__(*use_datetime=0*, *username=None*, *password=None*, *certChain=None*, *privateKey=None*, *checker=None*, *settings=None*, *ignoreAbruptClose=False*)
Create a new XMLRPCTransport.

An instance of this class can be passed to xmlrpclib.ServerProxy to use TLS with XML-RPC calls:

```
from tlslite import XMLRPCTransport
from xmlrpclib import ServerProxy

transport = XMLRPCTransport(user="alice", password="abra123")
server = ServerProxy("https://localhost", transport)
```

For client authentication, use one of these argument combinations:

- username, password (SRP)
- certChain, privateKey (certificate)

For server authentication, you can either rely on the implicit mutual authentication performed by SRP or you can do certificate-based server authentication with one of these argument combinations:

- x509Fingerprint

Certificate-based server authentication is compatible with SRP or certificate-based client authentication.

The constructor does not perform the TLS handshake itself, but simply stores these arguments for later. The handshake is performed only when this class needs to connect with the server. Thus you should be prepared to handle TLS-specific exceptions when calling methods of xmlrpclib.ServerProxy. See the client handshake functions in *TLSConnection* for details on which exceptions might be raised.

Parameters

- **username** (*str*) – SRP username. Requires the 'password' argument.
- **password** (*str*) – SRP password for mutual authentication. Requires the 'username' argument.
- **certChain** (X509CertChain) – Certificate chain for client authentication. Requires the 'privateKey' argument. Excludes the SRP arguments.
- **privateKey** (RSAKey) – Private key for client authentication. Requires the 'certChain' argument. Excludes the SRP arguments.
- **checker** (Checker) – Callable object called after handshaking to evaluate the connection and raise an Exception if necessary.
- **settings** (HandshakeSettings) – Various settings which can be used to control the ciphersuites, certificate types, and SSL/TLS versions offered by the client.
- **ignoreAbruptClose** (*bool*) – ignore the TLSAbruptCloseError on unexpected hangup.

conn_class_is_http = False

make_connection(*host*)
Make a connection to *host*. Reuse keepalive connections.

tlslite.utils package

Toolkit for crypto and other stuff.

Submodules

tlslite.utils.aes module

Abstract class for AES.

class tlslite.utils.aes.**AES**(*key*, *mode*, *IV*, *implementation*)
Bases: object

__init__(*key*, *mode*, *IV*, *implementation*)
Initialize self. See help(type(self)) for accurate signature.

decrypt(*ciphertext*)

encrypt(*plaintext*)

tlslite.utils.aesgcm module

class tlslite.utils.aesgcm.**AESGCM**(*key*, *implementation*, *rawAesEncrypt*)
Bases: object

AES-GCM implementation. Note: this implementation does not attempt to be side-channel resistant. It's also rather slow.

__init__(*key*, *implementation*, *rawAesEncrypt*)
Initialize self. See help(type(self)) for accurate signature.

open(*nonce*, *ciphertext*, *data*)
Decrypts and authenticates ciphertext using nonce and data. If the tag is valid, the plaintext is returned. If the tag is invalid, returns None.

seal(*nonce*, *plaintext*, *data*)
Encrypts and authenticates plaintext using nonce and data. Returns the ciphertext, consisting of the encrypted plaintext and tag concatenated.

tlslite.utils.asn1parser module

Abstract Syntax Notation One (ASN.1) parsing

class tlslite.utils.asn1parser.**ASN1Parser**(*bytes*)
Bases: object

Parser and storage of ASN.1 DER encoded objects.

Variables

- **length** (*int*) – length of the value of the tag
- **value** (*bytearray*) – literal value of the tag

__init__(*bytes*)
Create an object from bytes.

Parameters **bytes** (*bytearray*) – DER encoded ASN.1 object

getChild(*which*)
Return n-th child assuming that the object is a SEQUENCE.

Parameters **which** (*int*) – ordinal of the child to return

Return type *ASN1Parser*

Returns decoded child object

getChildBytes(*which*)
Return raw encoding of n-th child, assume self is a SEQUENCE

Parameters **which** (*int*) – ordinal of the child to return

Return type bytearray

Returns raw child object

getChildCount()
Return number of children, assuming that the object is a SEQUENCE.

Return type int

Returns number of children in the object

class tlslite.utils.asn1parser.**ASN1Type**(*tag_class*, *is_primitive*, *tag_id*)
Bases: object

Class that represents the ASN.1 type bit octet. Consists of a class (universal(0), application(1), context-specific(2) or private(3)), boolean value that indicates if a type is constructed or primitive and the ASN1 type itself.

Variables

- **field** – bit octet
- **tagClass** (*int*) – type's class

- **isPrimitive** (*int*) – equals to 0 if the type is primitive, 1 if not
- **tagId** (*int*) – ANS1 tag number

__init__(*tag_class*, *is_primitive*, *tag_id*)
Initialize self. See help(type(self)) for accurate signature.

tlslite.utils.chacha module

Pure Python implementation of ChaCha cipher

Implementation that follows RFC 7539 closely.

class tlslite.utils.chacha.**ChaCha**(*key*, *nonce*, *counter=0*, *rounds=20*)
Bases: object

Pure python implementation of ChaCha cipher

__init__(*key*, *nonce*, *counter=0*, *rounds=20*)
Set the initial state for the ChaCha cipher

static chacha_block(*key*, *counter*, *nonce*, *rounds*)
Generate a state of a single block

constants = [1634760805, 857760878, 2036477234, 1797285236]

decrypt(*ciphertext*)
Decrypt the data

classmethod double_round(*x*)
Perform two rounds of ChaCha cipher

encrypt(*plaintext*)
Encrypt the data

static quarter_round(*x*, *a*, *b*, *c*, *d*)
Perform a ChaCha quarter round

static rotl32(*v*, *c*)
Rotate left a 32 bit integer v by c bits

static word_to_bytearray(*state*)
Convert state to little endian bytestream

tlslite.utils.chacha20_poly1305 module

Pure Python implementation of ChaCha20/Poly1305 AEAD cipher

Implementation that follows RFC 7539 and draft-ietf-tls-chacha20-poly1305-00

class tlslite.utils.chacha20_poly1305.**CHACHA20_POLY1305**(*key*, *implementation*)
Bases: object

Pure python implementation of ChaCha20/Poly1305 AEAD cipher

__init__(*key*, *implementation*)
Set the initial state for the ChaCha20 AEAD

open(*nonce*, *ciphertext*, *data*)
Decrypts and authenticates ciphertext using nonce and data. If the tag is valid, the plaintext is returned. If the tag is invalid, returns None.

static pad16(*data*)
Return padding for the Associated Authenticated Data

static poly1305_key_gen(*key*, *nonce*)
Generate the key for the Poly1305 authenticator

seal(*nonce*, *plaintext*, *data*)
Encrypts and authenticates plaintext using nonce and data. Returns the ciphertext, consisting of the encrypted plaintext and tag concatenated.

tlslite.utils.cipherfactory module

Factory functions for symmetric cryptography.

tlslite.utils.cipherfactory.**createAES**(*key*, *IV*, *implList=None*)
Create a new AES object.

Parameters

- **key** (*str*) – A 16, 24, or 32 byte string.
- **IV** (*str*) – A 16 byte string

Return type tlslite.utils.AES

Returns An AES object.

tlslite.utils.cipherfactory.**createAESCCM**(*key*, *implList=None*)
Create a new AESCCM object.

Parameters **key** (*bytearray*) – A 16 or 32 byte byte array to serve as key.

Return type tlslite.utils.AESCCM

Returns An AESCCM object.

tlslite.utils.cipherfactory.**createAESCCM_8**(*key*, *implList=None*)
Create a new AESCCM object with truncated tag.

Parameters **key** (*bytearray*) – A 16 or 32 byte byte array to serve as key.

Return type tlslite.utils.AESCCM

Returns An AESCCM object.

tlslite.utils.cipherfactory.**createAESCTR**(*key*, *IV*, *implList=None*)
Create a new AESCTR object.

Parameters

- **key** (*str*) – A 16, 24, or 32 byte string.
- **IV** (*str*) – A 8 or 12 byte string

Return type tlslite.utils.AES

Returns An AES object.

tlslite.utils.cipherfactory.**createAESGCM**(*key*, *implList=None*)
Create a new AESGCM object.

Parameters **key** (*bytearray*) – A 16 or 32 byte byte array.

Return type tlslite.utils.AESGCM

Returns An AESGCM object.

tlslite.utils.cipherfactory.**createCHACHA20**(*key*, *implList=None*)
Create a new CHACHA20_POLY1305 object.

Parameters **key** (*bytearray*) – a 32 byte array to serve as key

Return type tlslite.utils.CHACHA20_POLY1305

Returns A ChaCha20/Poly1305 object

tlslite.utils.cipherfactory.**createRC4**(*key*, *IV*, *implList=None*)
Create a new RC4 object.

Parameters

- **key** (*str*) – A 16 to 32 byte string.
- **IV** (*object*) – Ignored, whatever it is.

Return type tlslite.utils.RC4

Returns An RC4 object.

tlslite.utils.cipherfactory.**createTripleDES**(*key*, *IV*, *implList=None*)
Create a new 3DES object.

Parameters

- **key** (*str*) – A 24 byte string.
- **IV** (*str*) – An 8 byte string

Return type tlslite.utils.TripleDES

Returns A 3DES object.

tlslite.utils.cipherfactory.**tripleDESPresent = True**
Inform if the 3DES algorithm is supported.

tlslite.utils.codec module

Classes for reading/writing binary data (such as TLS records).

exception tlslite.utils.codec.**BadCertificateError**
Bases: SyntaxError

Exception raised in case of bad certificate.

exception tlslite.utils.codec.**DecodeError**
Bases: SyntaxError

Exception raised in case of decoding errors.

class tlslite.utils.codec.**Parser**(*bytes*)
Bases: object

Parser for TLV and LV byte-based encodings.

Parser that can handle arbitrary byte-based encodings usually employed in Type-Length-Value or Length-Value binary encoding protocols like ASN.1 or TLS

Note: if the raw bytes don't match expected values (like trying to read a 4-byte integer from a 2-byte buffer), most methods will raise a DecodeError exception.

TODO: don't use an exception used by language parser to indicate errors in application code.

Variables

- **bytes** (*bytearray*) – data to be interpreted (buffer)
- **index** (*int*) – current position in the buffer
- **lengthCheck** (*int*) – size of struct being parsed
- **indexCheck** (*int*) – position at which the structure begins in buffer

__init__(*bytes*)
Bind raw bytes with parser.

Parameters **bytes** (*bytearray*) – bytes to be parsed/interpreted

atLengthCheck()
Check if there is data in structure left for parsing.

Returns True if the whole structure was parsed, False if there is some data left.

Will raise an exception if overflow occured (amount of data read was greater than expected size)

get(*length*)
Read a single big-endian integer value encoded in 'length' bytes.

Parameters **length** (*int*) – number of bytes in which the value is encoded in

Return type int

getFixBytes(*lengthBytes*)
Read a string of bytes encoded in 'lengthBytes' bytes.

Parameters **lengthBytes** (*int*) – number of bytes to return

Return type bytearray

getFixList(*length*, *lengthList*)
Read a list of static length with same-sized ints.

Parameters

- **length** (*int*) – size in bytes of a single element in list
- **lengthList** (*int*) – number of elements in list

Return type list of int

getRemainingLength()
Return amount of data remaining in struct being parsed.

getVarBytes(*lengthLength*)
Read a variable length string with a fixed length.

see Writer.add_var_bytes() for an inverse of this method

Parameters **lengthLength** (*int*) – number of bytes in which the length of the string is encoded in

Return type bytearray

getVarList(*length*, *lengthLength*)
Read a variable length list of same-sized integers.

Parameters

- **length** (*int*) – size in bytes of a single element
- **lengthLength** (*int*) – size of the encoded length of the list

Return type list of int

getVarTupleList(*elemLength*, *elemNum*, *lengthLength*)
Read a variable length list of same sized tuples.

Parameters

- **elemLength** (*int*) – length in bytes of single tuple element
- **elemNum** (*int*) – number of elements in tuple
- **lengthLength** (*int*) – length in bytes of the list length variable

Return type list of tuple of int

setLengthCheck(*length*)
Set length of struct and start a length check for parsing.

Parameters **length** (*int*) – expected size of parsed struct in bytes

skip_bytes(*length*)
Move the internal pointer ahead length bytes.

startLengthCheck(*lengthLength*)
Read length of struct and start a length check for parsing.

Parameters **lengthLength** (*int*) – number of bytes in which the length is encoded

stopLengthCheck()
Stop struct parsing, verify that no under- or overflow occurred.

In case the expected length was mismatched with actual length of processed data, raises an exception.

class tlslite.utils.codec.**Writer**
Bases: object

Serialisation helper for complex byte-based structures.

__init__()
Initialise the serializer with no data.

add(*x*, *length*)
Add a single positive integer value x, encode it in length bytes

Encode positive integer x in big-endian format using length bytes, add to the internal buffer.

Parameters

- **x** (*int*) – value to encode
- **length** (*int*) – number of bytes to use for encoding the value

addFixSeq(*seq*, *length*)
Add a list of items, encode every item in length bytes

Uses the unbounded iterable seq to produce items, each of which is then encoded to length bytes

Parameters

- **seq** (*iterable of int*) – list of positive integers to encode
- **length** (*int*) – number of bytes to which encode every element

addFour(*val*)
Add a four-byte wide element to buffer, see add().

addOne(*val*)
Add a single-byte wide element to buffer, see add().

addThree(*val*)
Add a three-byte wide element to buffer, see add().

addTwo(*val*)
Add a double-byte wide element to buffer, see add().

addVarSeq(*seq*, *length*, *lengthLength*)
Add a bounded list of same-sized values

Create a list of specific length with all items being of the same size

Parameters

- **seq** (*list of int*) – list of positive integers to encode
- **length** (*int*) – amount of bytes in which to encode every item
- **lengthLength** (*int*) – amount of bytes in which to encode the overall length of the array

addVarTupleSeq(*seq*, *length*, *lengthLength*)
Add a variable length list of same-sized element tuples.

Note that all tuples must have the same size.

Inverse of Parser.getVarTupleList()

Parameters

- **seq** (*enumerable*) – list of tuples
- **length** (*int*) – length of single element in tuple
- **lengthLength** (*int*) – length in bytes of overall length field

add_var_bytes(*data*, *length_length*)
Add a variable length array of bytes.

Inverse of Parser.getVarBytes()

Parameters

- **data** (*bytes*) – bytes to add to the buffer
- **length_length** (*int*) – size of the field to represent the length of the data string

tlslite.utils.codec.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.compat module

Miscellaneous functions to mask Python version differences.

tlslite.utils.compat.**a2b_base64**(*s*)

tlslite.utils.compat.**a2b_hex**(*s*)

tlslite.utils.compat.**b2a_base64**(*b*)

tlslite.utils.compat.**b2a_hex**(*b*)

tlslite.utils.compat.**bit_length**(*val*)
Return number of bits necessary to represent an integer.

tlslite.utils.compat.**byte_length**(*val*)
Return number of bytes necessary to represent an integer.

tlslite.utils.compat.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.compat.**compat26Str**(*x*)

tlslite.utils.compat.**compatAscii2Bytes**(*val*)
Convert ASCII string to bytes.

tlslite.utils.compat.**compatHMAC**(*x*)
Convert bytes-like input to format acceptable for HMAC.

tlslite.utils.compat.**compatLong**(*num*)

tlslite.utils.compat.**formatExceptionTrace**(*e*)
Return exception information formatted as string

tlslite.utils.compat.**int_to_bytes**(*val*, *length=None*, *byteorder='big'*)
Return number converted to bytes

tlslite.utils.compat.**raw_input**(*s*)

tlslite.utils.compat.**readStdinBinary**()

tlslite.utils.compat.**remove_whitespace**(*text*)
Removes all whitespace from passed in string

tlslite.utils.compat.**time_stamp**()
Returns system time as a float

tlslite.utils.constanttime module

Various constant time functions for processing sensitive data

tlslite.utils.constanttime.**ct_check_cbc_mac_and_pad**(*data*, *mac*, *seqnumBytes*, *contentType*, *version*, *block_size=16*)
Check CBC cipher HMAC and padding. Close to constant time.

Parameters

- **data** (*bytearray*) – data with HMAC value to test and padding

- **mac** (*hashlib mac*) – empty HMAC, initialised with a key
- **seqnumBytes** (*bytearray*) – TLS sequence number, used as input to HMAC
- **contentType** (*int*) – a single byte, used as input to HMAC
- **version** (*tuple of int*) – a tuple of two ints, used as input to HMAC and to guide checking of padding

Return type boolean

Returns True if MAC and pad is ok, False otherwise

tlslite.utils.constanttime.**ct_eq_u32**(*val_a*, *val_b*)
Return 1 if val_a == val_b, 0 otherwise. Constant time.

Parameters

- **val_a** (*int*) – an unsigned integer representable as a 32 bit value
- **val_b** (*int*) – an unsigned integer representable as a 32 bit value

Return type int

tlslite.utils.constanttime.**ct_gt_u32**(*val_a*, *val_b*)
Return 1 if val_a > val_b, 0 otherwise. Constant time.

Parameters

- **val_a** (*int*) – an unsigned integer representable as a 32 bit value
- **val_b** (*int*) – an unsigned integer representable as a 32 bit value

Return type int

tlslite.utils.constanttime.**ct_isnonzero_u32**(*val*)
Returns 1 if val is != 0, 0 otherwise. Constant time.

Parameters **val** (*int*) – an unsigned integer representable as a 32 bit value

Return type int

tlslite.utils.constanttime.**ct_le_u32**(*val_a*, *val_b*)
Return 1 if val_a <= val_b, 0 otherwise. Constant time.

Parameters

- **val_a** (*int*) – an unsigned integer representable as a 32 bit value
- **val_b** (*int*) – an unsigned integer representable as a 32 bit value

Return type int

tlslite.utils.constanttime.**ct_lsb_prop_u8**(*val*)
Propagate LSB to all 8 bits of the returned byte. Constant time.

tlslite.utils.constanttime.**ct_lt_u32**(*val_a*, *val_b*)
Returns 1 if val_a < val_b, 0 otherwise. Constant time.

Parameters

- **val_a** (*int*) – an unsigned integer representable as a 32 bit value
- **val_b** (*int*) – an unsigned integer representable as a 32 bit value

Return type int

tlslite.utils.constanttime.**ct_neq_u32**(*val_a*, *val_b*)
Return 1 if val_a != val_b, 0 otherwise. Constant time.

Parameters

- **val_a** (*int*) – an unsigned integer representable as a 32 bit value
- **val_b** (*int*) – an unsigned integer representable as a 32 bit value

Return type int

tlslite.utils.cryptomath module

cryptomath module

This module has basic math/crypto code.

tlslite.utils.cryptomath.**HKDF_expand**(*PRK*, *info*, *L*, *algorithm*)

tlslite.utils.cryptomath.**HKDF_expand_label**(*secret*, *label*, *hashValue*, *length*, *algorithm*)
TLS1.3 key derivation function (HKDF-Expand-Label).

Parameters

- **secret** (*bytearray*) – the key from which to derive the keying material
- **label** (*bytearray*) – label used to differentiate the keying materials
- **hashValue** (*bytearray*) – bytes used to "salt" the produced keying material
- **length** (*int*) – number of bytes to produce
- **algorithm** (*str*) – name of the secure hash algorithm used as the basis of the HKDF

Return type bytearray

tlslite.utils.cryptomath.**HMAC_MD5**(*k*, *b*)

tlslite.utils.cryptomath.**HMAC_SHA1**(*k*, *b*)

tlslite.utils.cryptomath.**HMAC_SHA256**(*k*, *b*)

tlslite.utils.cryptomath.**HMAC_SHA384**(*k*, *b*)

tlslite.utils.cryptomath.**MD5**(*b*)
Return a MD5 digest of data

tlslite.utils.cryptomath.**SHA1**(*b*)
Return a SHA1 digest of data

tlslite.utils.cryptomath.**bytesToNumber**(*b*, *endian='big'*)
Convert a number stored in bytearray to an integer.

By default assumes big-endian encoding of the number.

tlslite.utils.cryptomath.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.cryptomath.**derive_secret**(*secret*, *label*, *handshake_hashes*, *algorithm*)
TLS1.3 key derivation function (Derive-Secret).

Parameters

- **secret** (*bytearray*) – secret key used to derive the keying material
- **label** (*bytearray*) – label used to differentiate they keying materials
- **handshake_hashes** (HandshakeHashes) – hashes of the handshake messages or *None* if no handshake transcript is to be used for derivation of keying material
- **algorithm** (*str*) – name of the secure hash algorithm used as the basis of the HKDF algorithm - governs how much keying material will be generated

Return type bytearray

tlslite.utils.cryptomath.**divceil**(*divident*, *divisor*)
Integer division with rounding up

tlslite.utils.cryptomath.**gcd**(*a*, *b*)

tlslite.utils.cryptomath.**getRandomBytes**(*howMany*)

tlslite.utils.cryptomath.**getRandomNumber**(*low*, *high*)

tlslite.utils.cryptomath.**getRandomPrime**(*bits*, *display=False*)
Generate a random prime number of a given size.

the number will be 'bits' bits long (i.e. generated number will be larger than *(2^(bits-1) * 3) / 2* but smaller than 2^bits.

tlslite.utils.cryptomath.**getRandomSafePrime**(*bits*, *display=False*)
Generate a random safe prime.

Will generate a prime *bits* bits long (see getRandomPrime) such that the (p-1)/2 will also be prime.

tlslite.utils.cryptomath.**invMod**(*a*, *b*)
Return inverse of a mod b, zero if none.

tlslite.utils.cryptomath.**isPrime**(*n, iterations=5, display=False, sieve=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]*)

tlslite.utils.cryptomath.**lcm**(*a*, *b*)

tlslite.utils.cryptomath.**makeSieve**(*n*)

tlslite.utils.cryptomath.**mpiToNumber**(*mpi*)
Convert a MPI (OpenSSL bignum string) to an integer.

tlslite.utils.cryptomath.**numberToByteArray**(*n*, *howManyBytes=None*, *endian='big'*)
Convert an integer into a bytearray, zero-pad to howManyBytes.

The returned bytearray may be smaller than howManyBytes, but will not be larger. The returned bytearray will contain a big- or little-endian encoding of the input integer (n). Big endian encoding is used by default.

tlslite.utils.cryptomath.**numberToMPI**(*n*)

tlslite.utils.cryptomath.**secureHMAC**(*k*, *b*, *algorithm*)
Return a HMAC using *b* and *k* using *algorithm*

tlslite.utils.cryptomath.**secureHash**(*data*, *algorithm*)
Return a digest of *data* using *algorithm*

tlslite.utils.datefuncs module

tlslite.utils.datefuncs.**createDateClass**(*year*, *month*, *day*, *hour*, *minute*, *second*)

tlslite.utils.datefuncs.**getHoursFromNow**(*hours*)

tlslite.utils.datefuncs.**getMinutesFromNow**(*minutes*)

tlslite.utils.datefuncs.**getNow**()

tlslite.utils.datefuncs.**isDateClassBefore**(*d1*, *d2*)

tlslite.utils.datefuncs.**isDateClassExpired**(*d*)

tlslite.utils.datefuncs.**parseDateClass**(*s*)

tlslite.utils.datefuncs.**printDateClass**(*d*)

tlslite.utils.deprecations module

Methods for deprecating old names for arguments or attributes.

tlslite.utils.deprecations.**deprecated_attrs**(*names*, *warn="Attribute '{old_name}' is deprecated, please use '{new_name}'"*)
Decorator to deprecate all specified attributes in class.

Translates all names in *names* to use new names and emits warnings if the translation was necessary.

Note: uses metaclass magic so is incompatible with other metaclass uses

Parameters

- **names** (*dict*) – dictionary with paris of new_name: old_name that will be used to translate the calls
- **warn** (*str*) – DeprecationWarning format string for informing the user what is the current parameter name, uses 'old_name' for the deprecated keyword name and 'new_name' for the current one. Example: "Old name: {old_name}, use {new_name} instead".

tlslite.utils.deprecations.**deprecated_class_name**(*old_name*, *warn="Class name '{old_name}' is deprecated, please use '{new_name}'"*)
Class decorator to deprecate a use of class.

Parameters

- **old_name** (*str*) – the deprecated name that will be registered, but will raise warnings if used.

- **warn** (*str*) – DeprecationWarning format string for informing the user what is the current class name, uses 'old_name' for the deprecated keyword name and the 'new_name' for the current one. Example: "Old name: {old_nam}, use '{new_name}' instead".

tlslite.utils.deprecations.**deprecated_instance_attrs**(*names*, *warn="Attribute '{old_name}' is deprecated, please use '{new_name}'"*)

Decorator to deprecate class instance attributes.

Translates all names in *names* to use new names and emits warnings if the translation was necessary. Does apply only to instance variables and attributes (won't modify behaviour of class variables, static methods, etc.

Parameters

- **names** (*dict*) – dictionary with paris of new_name: old_name that will be used to translate the calls
- **warn** (*str*) – DeprecationWarning format string for informing the user what is the current parameter name, uses 'old_name' for the deprecated keyword name and 'new_name' for the current one. Example: "Old name: {old_name}, use {new_name} instead".

tlslite.utils.deprecations.**deprecated_method**(*message*)

Decorator for deprecating methods.

Parameters **message** (*ste*) – The message you want to display.

tlslite.utils.deprecations.**deprecated_params**(*names*, *warn="Param name '{old_name}' is deprecated, please use '{new_name}'"*)

Decorator to translate obsolete names and warn about their use.

Parameters

- **names** (*dict*) – dictionary with pairs of new_name: old_name that will be used for translating obsolete param names to new names
- **warn** (*str*) – DeprecationWarning format string for informing the user what is the current parameter name, uses 'old_name' for the deprecated keyword name and 'new_name' for the current one. Example: "Old name: {old_name}, use {new_name} instead".

tlslite.utils.dns_utils module

Utilities for handling DNS hostnames

tlslite.utils.dns_utils.**is_valid_hostname**(*hostname*)

Check if the parameter is a valid hostname.

Parameters **hostname** (*str or bytearray*) – string to check

Return type boolean

tlslite.utils.ecc module

Methods for dealing with ECC points

tlslite.utils.ecc.**decodeX962Point**(*data*, *curve=NIST256p*)

Decode a point from a X9.62 encoding

tlslite.utils.ecc.**encodeX962Point**(*point*)

Encode a point in X9.62 format

tlslite.utils.ecc.**getCurveByName**(*curveName*)
Return curve identified by curveName

tlslite.utils.ecc.**getPointByteSize**(*point*)
Convert the point or curve bit size to bytes

tlslite.utils.ecdsakey module

Abstract class for ECDSA.

class tlslite.utils.ecdsakey.**ECDSAKey**(*public_key*, *private_key*)
Bases: object

This is an abstract base class for ECDSA keys.

Particular implementations of ECDSA keys, such as Python_ECDSAKey ... more coming inherit from this.

To create or parse an ECDSA key, don't use one of these classes directly. Instead, use the factory functions in *keyfactory*.

__init__(*public_key*, *private_key*)
Create a new ECDSA key.

If public_key or private_key are passed in, the new key will be initialized.

Parameters

- **public_key** – ECDSA public key.
- **private_key** – ECDSA private key.

acceptsPassword()
Return True if the write() method accepts a password for use in encrypting the private key.

Return type bool

static generate(*bits*)
Generate a new key with the specified curve.

Return type ECDSAKey

hasPrivateKey()
Return whether or not this key has a private component.

Return type bool

hashAndSign(*bytes*, *rsaScheme=None*, *hAlg='sha1'*, *sLen=None*)
Hash and sign the passed-in bytes.

This requires the key to have a private component. It performs a signature on the passed-in data with selected hash algorithm.

Parameters

- **bytes** (*bytes-like object*) – The value which will be hashed and signed.
- **rsaScheme** (*str*) – Ignored, present for API compatibility with RSA
- **hAlg** (*str*) – The hash algorithm that will be used to hash data
- **sLen** (*int*) – Ignored, present for API compatibility with RSA

Return type bytearray

Returns An ECDSA signature on the passed-in data.

hashAndVerify(*sigBytes*, *bytes*, *rsaScheme=None*, *hAlg='sha1'*, *sLen=None*)
Hash and verify the passed-in bytes with the signature.

This verifies an ECDSA signature on the passed-in data with selected hash algorithm.

Parameters

- **sigBytes** (*bytearray*) – An ECDSA signature, DER encoded.
- **bytes** (*str or bytearray*) – The value which will be hashed and verified.
- **rsaScheme** (*str*) – Ignored, present for API compatibility with RSA
- **hAlg** (*str*) – The hash algorithm that will be used
- **sLen** (*int*) – Ignored, present for API compatibility with RSA

Return type bool

Returns Whether the signature matches the passed-in data.

sign(*bytes*, *padding=None*, *hashAlg='sha1'*, *saltLen=None*)
Sign the passed-in bytes.

This requires the key to have a private component. It performs an ECDSA signature on the passed-in data.

Parameters

- **bytes** (*bytearray*) – The value which will be signed (generally a binary encoding of hash output.
- **padding** (*str*) – Ignored, present for API compatibility with RSA
- **hashAlg** (*str*) – name of hash that was used for calculating the bytes
- **saltLen** (*int*) – Ignored, present for API compatibility with RSA

Return type bytearray

Returns An ECDSA signature on the passed-in data.

verify(*sigBytes*, *bytes*, *padding=None*, *hashAlg=None*, *saltLen=None*)
Verify the passed-in bytes with the signature.

This verifies a PKCS1 signature on the passed-in data.

Parameters

- **sigBytes** (*bytearray*) – A PKCS1 signature.
- **bytes** (*bytearray*) – The value which will be verified.
- **padding** (*str*) – Ignored, present for API compatibility with RSA

Return type bool

Returns Whether the signature matches the passed-in data.

write(*password=None*)
Return a string containing the key.

Return type str

Returns A string describing the key, in whichever format (PEM) is native to the implementation.

tlslite.utils.keyfactory module

Factory functions for asymmetric cryptography.

tlslite.utils.keyfactory.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.keyfactory.**generateRSAKey**(*bits, implementations=['openssl', 'python']*)
Generate an RSA key with the specified bit length.

Parameters **bits** (*int*) – Desired bit length of the new key's modulus.

Return type *RSAKey*

Returns A new RSA private key.

tlslite.utils.keyfactory.**parseAsPublicKey**(*s*)
Parse a PEM-formatted public key.

Parameters **s** (*str*) – A string containing a PEM-encoded public or private key.

Return type *RSAKey*

Returns An RSA public key.

Raises **SyntaxError** – If the key is not properly formatted.

tlslite.utils.keyfactory.**parsePEMKey**(*s, private=False, public=False, passwordCallback=None, implementations=['openssl', 'python']*)
Parse a PEM-format key.

The PEM format is used by OpenSSL and other tools. The format is typically used to store both the public and private components of a key. For example:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDYscuoMzsGmW0pAYsmyHltxB2TdwHS0dImfjCMfaSDkfLdZY5+
dOWORVns9etWnr194mSGA1F0Pls/VJW8+cX9+3vtJV8zSdANPYUoQf0TP7VlJxkH
dSRkUbEoz5bAAs/+970uos7n7iXQIni+3erUTdYEk2iWnMBjTljfgbK/dQIDAQAB
AoGAJHoJZk75aKr7DSQNYIHuruOMdv5ZeDuJvKERWxTrVJqE32/xBKh42/IgqRrc
esBN9ZregRCd7YtxoL+EVUNWaJNVx2mNmezEznrc9zhcYUrgeaVdFO2yBF1889zO
gCOVwrO8uDgeyj6IKa25H6c1N13ih/o7ZzEgWbGG+ylU1yECQQDv4ZSJ4EjSh/Fl
aHdz3wbBa/HKGTjC8iRy476Cyg2Fm8MZUe9Yy3udOrb5ZnS2MTpIXt5AF3h2TfYV
VoFXIorjAkEA50FcJmzT8sNMrPaV8vn+9W2Lu4U7C+K/O2g1iXMaZms5PC5zV5aV
CKXZWUX1fq2RaOzlbQrpgiolhXpeh8FjxwJBAOFHzSQfSsTNfttp3KUpU0LbiVvv
i+spVSnA0O4rq79KpVNmK44Mq67hsW1P11QzrzTAQ6GVaUBRv0YS061td1kCQHnP
wtN2tboFR6lABkJDjxoGRvlSt4SOPr7zKGgrWjeiuTZLHXSAnCY+/hr5L9Q3ZwXG
6x6iBdgLjVIe4BZQNtcCQQDXGv/gWinCNTN3MPWfTW/RGzuMYVmyBFais0/VrgdH
h1dLpztmpQqfyH/zrBXQ9qL/zR4ojS6XYneO/U18WpEe
-----END RSA PRIVATE KEY-----
```

To generate a key like this with OpenSSL, run:

```
openssl genrsa 2048 > key.pem
```

This format also supports password-encrypted private keys. TLS Lite can only handle password-encrypted private keys when OpenSSL and M2Crypto are installed. In this case, passwordCallback will be invoked to query the user for the password.

Parameters

- **s** (*str*) – A string containing a PEM-encoded public or private key.
- **private** (*bool*) – If True, a SyntaxError will be raised if the private key component is not present.
- **public** (*bool*) – If True, the private key component (if present) will be discarded, so this function will always return a public key.
- **passwordCallback** (*callable*) – This function will be called, with no arguments, if the PEM-encoded private key is password-encrypted. The callback should return the password string. If the password is incorrect, SyntaxError will be raised. If no callback is passed and the key is password-encrypted, a prompt will be displayed at the console.

Return type *RSAKey*

Returns An RSA key.

Raises **SyntaxError** – If the key is not properly formatted.

tlslite.utils.keyfactory.**parsePrivateKey**(*s*)

Parse a PEM-formatted private key.

Parameters **s** (*str*) – A string containing a PEM-encoded private key.

Return type *RSAKey*

Returns An RSA private key.

Raises **SyntaxError** – If the key is not properly formatted.

tlslite.utils.lists module

Helper functions for handling lists

tlslite.utils.lists.**getFirstMatching**(*values*, *matches*)

Return the first element in `values` that is also in `matches`.

Return None if values is None, empty or no element in values is also in matches.

Parameters

- **values** (*collections.abc.Iterable*) – list of items to look through, can be None
- **matches** (*collections.abc.Container*) – list of items to check against

tlslite.utils.lists.**to_str_delimiter**(*values*, *delim=', '*, *last_delim=' or '*)

Format the list as a human readable string.

Will format the list as a human readable enumeration, separated by commas (changable with *delim*) with last value separated with "or" (changable with *last_delim*).

Parameters

- **values** (*collections.abc.Iterable*) – list of items to concatenate
- **delim** (*str*) – primary delimiter for objects, comma by default

- **last_delim** (*str*) – delimiter for last object in list

Return type str

tlslite.utils.openssl_aes module

OpenSSL/M2Crypto AES implementation.

tlslite.utils.openssl_aes.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.openssl_rc4 module

OpenSSL/M2Crypto RC4 implementation.

tlslite.utils.openssl_rc4.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.openssl_rsakey module

OpenSSL/M2Crypto RSA implementation.

tlslite.utils.openssl_rsakey.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.openssl_rsakey.**password_callback**(*v*, *prompt1='Enter private key passphrase:'*, *prompt2='Verify passphrase:'*)

tlslite.utils.openssl_tripledes module

OpenSSL/M2Crypto 3DES implementation.

tlslite.utils.openssl_tripledes.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.pem module

tlslite.utils.pem.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.pem.**dePem**(*s*, *name*)
Decode a PEM string into a bytearray of its payload.

The input must contain an appropriate PEM prefix and postfix based on the input name string, e.g. for name="CERTIFICATE":

```
-----BEGIN CERTIFICATE-----
MIIBXDCCAUSgAwIBAgIBADANBgkqhkiG9w0BAQUFADAPMQ0wCwYDVQQDEwRUQUNL
...
KoZIhvcNAQEFBQADAwA5kw==
-----END CERTIFICATE-----
```

The first such PEM block in the input will be found, and its payload will be base64 decoded and returned.

tlslite.utils.pem.**dePemList**(*s*, *name*)
Decode a sequence of PEM blocks into a list of bytearrays.

The input must contain any number of PEM blocks, each with the appropriate PEM prefix and postfix based on the input name string, e.g. for name="TACK BREAK SIG". Arbitrary text can appear between and before and after the PEM blocks. For example:

```
Created by TACK.py 0.9.3 Created at 2012-02-01T00:30:10Z
-----BEGIN TACK BREAK SIG-----
ATKhrz5C6JHJW8BF5fLVrnQss6JnWVyEaC0p89LNhKPswvcC9/s6+vWLd9snYTUv
YMEBdw69PUP8JB4AdqA3K6Ap0Fgd9SSTOECeAKOUAym8zcYaXUwpk0+WuPYa7Zmm
SkbOlK4ywqt+amhWbg9txSGUwFO5tWUHT3QrnRlE/e3PeNFXLx5Bckg=
-----END TACK BREAK SIG-----
```

(continues on next page)

(continued from previous page)

```
Created by TACK.py 0.9.3 Created at 2012-02-01T00:30:11Z
-----BEGIN TACK BREAK SIG-----
ATKhrz5C6JHJW8BF5fLVrnQss6JnWVyEaC0p89LNhKPswvcC9/s6+vWLd9snYTUv
YMEBdw69PUP8JB4AdqA3K6BVCWfcjN36lx6JwxmZQncS6sww7DecFO/qjSePCxwM
+kdDqX/9/183nmjx6bf0ewhPXkA0nVXsDYZaydN8rJU1GaMlnjcIYxY=
-----END TACK BREAK SIG-----
```

All such PEM blocks will be found, decoded, and return in an ordered list of bytearrays, which may have zero elements if not PEM blocks are found.

tlslite.utils.pem.**pem**(*b*, *name*)

Encode a payload bytearray into a PEM string.

The input will be base64 encoded, then wrapped in a PEM prefix/postfix based on the name string, e.g. for name="CERTIFICATE":

```
-----BEGIN CERTIFICATE-----
MIIBXDCCAUSgAwIBAgIBADANBgkqhkiG9w0BAQUFADAPMQ0wCwYDVQQDEwRUQUNL
...
KoZIhvcNAQEFBQADAwA5kw==
-----END CERTIFICATE-----
```

tlslite.utils.pem.**pemSniff**(*inStr*, *name*)

tlslite.utils.poly1305 module

Implementation of Poly1305 authenticator for RFC 7539

class tlslite.utils.poly1305.**Poly1305**(*key*)

Bases: object

Poly1305 authenticator

P = 1361129467683753853853498429727072845819

__init__(*key*)

Set the authenticator key

create_tag(*data*)

Calculate authentication tag for data

static le_bytes_to_num(*data*)

Convert a number from little endian byte format

static num_to_16_le_bytes(*num*)

Convert number to 16 bytes in little endian format

tlslite.utils.pycrypto_aes module

PyCrypto AES implementation.

tlslite.utils.pycrypto_aes.**bytes_to_int**()

Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.pycrypto_aesgcm module

PyCrypto AES-GCM implementation.

`tlslite.utils.pycrypto_aesgcm.`**`bytes_to_int`**`()`
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.pycrypto_rc4 module

PyCrypto RC4 implementation.

`tlslite.utils.pycrypto_rc4.`**`bytes_to_int`**`()`
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.pycrypto_rsakey module

PyCrypto RSA implementation.

`tlslite.utils.pycrypto_rsakey.`**`bytes_to_int`**`()`
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.pycrypto_tripledes module

PyCrypto 3DES implementation.

tlslite.utils.pycrypto_tripledes.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.python_aes module

Pure-Python AES implementation.

tlslite.utils.python_aes.**new**(*key*, *mode*, *IV*)

class tlslite.utils.python_aes.**Python_AES**(*key*, *mode*, *IV*)
Bases: *tlslite.utils.aes.AES*

__init__(*key*, *mode*, *IV*)
Initialize self. See help(type(self)) for accurate signature.

decrypt(*ciphertext*)

encrypt(*plaintext*)

tlslite.utils.python_aesgcm module

Pure-Python AES-GCM implementation.

tlslite.utils.python_aesgcm.**new**(*key*)

tlslite.utils.python_chacha20_poly1305 module

Pure-Python ChaCha20/Poly1305 implementation.

tlslite.utils.python_chacha20_poly1305.**new**(*key*)
Return an AEAD cipher implementation

tlslite.utils.python_rc4 module

Pure-Python RC4 implementation.

class tlslite.utils.python_rc4.**Python_RC4**(*keyBytes*)
Bases: *tlslite.utils.rc4.RC4*

__init__(*keyBytes*)
Initialize self. See help(type(self)) for accurate signature.

decrypt(*ciphertext*)

encrypt(*plaintextBytes*)

tlslite.utils.python_rc4.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.python_rc4.**new**(*key*)

tlslite.utils.python_rsakey module

Pure-Python RSA implementation.

class tlslite.utils.python_rsakey.**Python_RSAKey**(*n=0*, *e=0*, *d=0*, *p=0*, *q=0*, *dP=0*, *dQ=0*, *qInv=0*, *key_type='rsa'*)
Bases: *tlslite.utils.rsakey.RSAKey*

__init__(*n=0*, *e=0*, *d=0*, *p=0*, *q=0*, *dP=0*, *dQ=0*, *qInv=0*, *key_type='rsa'*)
Initialise key directly from integers.

see also generate() and parsePEM().

acceptsPassword()
Does it support encrypted key files.

static generate(*bits*, *key_type='rsa'*)
Generate a private key with modulus 'bits' bit big.

key_type can be "rsa" for a universal rsaEncryption key or "rsa-pss" for a key that can be used only for RSASSA-PSS.

hasPrivateKey()
Does the key has the associated private key (True) or is it only the public part (False).

static parsePEM(*data*, *password_callback=None*)
Parse a string containing a PEM-encoded <privateKey>.

tlslite.utils.python_rsakey.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.rc4 module

Abstract class for RC4.

class tlslite.utils.rc4.**RC4**(*keyBytes*, *implementation*)
Bases: object

__init__(*keyBytes*, *implementation*)
Initialize self. See help(type(self)) for accurate signature.

decrypt(*ciphertext*)

encrypt(*plaintext*)

tlslite.utils.rijndael module

A pure python (slow) implementation of rijndael with a decent interface

To include -

from rijndael import Rijndael

To do a key setup -

r = Rijndael(key, block_size = 16)

key must be a string of length 16, 24, or 32 blocksize must be 16, 24, or 32. Default is 16

To use -

ciphertext = r.encrypt(plaintext) plaintext = r.decrypt(ciphertext)

If any strings are of the wrong length a ValueError is thrown

class tlslite.utils.rijndael.**Rijndael**(*key*, *block_size=16*)
Bases: object

Implementation of the AES (formely known as Rijndael) block cipher.

Supports key sizes of 128, 192 and 256 bits as well as the non standard block sizes of 192 and 256 bits (the standard 128 bit block size is the default).

Note: this is a pure python stright-forward implementation thus it is vulnerable against majority, if not all possible side channel attacks.

Can process data just one block at a time, does not perform block cipher chaining or plaintext padding.

Ival int block_size the size of the encrypted blocks, in bytes

Ival list Ke key schedule for encryption

Ival list Kd key schedule for decryption

__init__(*key*, *block_size=16*)
Initialise the object, derive keys for encryption and decryption.

decrypt(*ciphertext*)
Decrypt a block of ciphertext.

encrypt(*plaintext*)
Encrypt a single block of plaintext.

tlslite.utils.rijndael.**decrypt**(*key*, *block*)

tlslite.utils.rijndael.**encrypt**(*key*, *block*)

tlslite.utils.rijndael.**rijndael**
alias of *tlslite.utils.rijndael.Rijndael*

tlslite.utils.rijndael.**test**()

tlslite.utils.rsakey module

Abstract class for RSA.

class tlslite.utils.rsakey.**RSAKey**(*n=0*, *e=0*, *key_type='rsa'*)
Bases: object

This is an abstract base class for RSA keys.

Particular implementations of RSA keys, such as OpenSSL_RSAKey, *Python_RSAKey*, and PyCrypto_RSAKey, inherit from this.

To create or parse an RSA key, don't use one of these classes directly. Instead, use the factory functions in *keyfactory*.

EMSA_PSS_encode(*mHash*, *emBits*, *hAlg*, *sLen=0*)
Encode the passed in message

This encodes the message using selected hash algorithm

Parameters

- **mHash** (*bytearray*) – Hash of message to be encoded
- **emBits** (*int*) – maximal length of returned EM
- **hAlg** (*str*) – hash algorithm to be used
- **sLen** (*int*) – length of salt

EMSA_PSS_verify(*mHash*, *EM*, *emBits*, *hAlg*, *sLen=0*)
Verify signature in passed in encoded message

This verifies the signature in encoded message

Parameters

- **mHash** (*bytes-like object*) – Hash of the original not signed message
- **EM** (*bytes-like object*) – Encoded message
- **emBits** (*int*) – Length of the encoded message in bits
- **hAlg** (*str*) – hash algorithm to be used
- **sLen** (*int*) – Length of salt

MGF1(*mgfSeed*, *maskLen*, *hAlg*)
Generate mask from passed-in seed.

This generates mask based on passed-in seed and output maskLen.

Parameters

- **mgfSeed** (*bytearray*) – Seed from which mask will be generated.
- **maskLen** (*int*) – Wished length of the mask, in octets

Return type bytearray

Returns Mask

RSASSA_PSS_sign(*mHash*, *hAlg*, *sLen=0*)
"Sign the passed in message

This signs the message using selected hash algorithm

Parameters

- **mHash** (*bytes-like object*) – Hash of message to be signed
- **hAlg** (*str*) – hash algorithm to be used
- **sLen** (*int*) – length of salt

RSASSA_PSS_verify(*mHash*, *S*, *hAlg*, *sLen=0*)
Verify the signature in passed in message

This verifies the signature in the signed message

Parameters

- **mHash** (*bytes-like object*) – Hash of original message
- **S** (*bytes-like object*) – Signed message
- **hAlg** (*str*) – Hash algorithm to be used
- **sLen** (*int*) – Length of salt

__init__(*n=0*, *e=0*, *key_type='rsa'*)
Create a new RSA key.

If n and e are passed in, the new key will be initialized.

Parameters

- **n** (*int*) – RSA modulus.
- **e** (*int*) – RSA public exponent.
- **key_type** (*str*) – type of the RSA key, "rsa" for rsaEncryption (universal, able to perform all operations) or "rsa-pss" for a RSASSA-PSS key (able to perform only RSA-PSS signature verification and creation)

acceptsPassword()
Return True if the write() method accepts a password for use in encrypting the private key.

Return type bool

classmethod addPKCS1Prefix(*data*, *hashName*)
Add the PKCS#1 v1.5 algorithm identifier prefix to hash bytes

classmethod addPKCS1SHA1Prefix(*hashBytes*, *withNULL=True*)
Add PKCS#1 v1.5 algorithm identifier prefix to SHA1 hash bytes

decrypt(*encBytes*)
Decrypt the passed-in bytes.

This requires the key to have a private component. It performs PKCS1 decryption of the passed-in data.

Parameters **encBytes** (*bytes-like object*) – The value which will be decrypted.

Return type bytearray or None

Returns A PKCS1 decryption of the passed-in data or None if the data is not properly formatted.

encrypt(*bytes*)
Encrypt the passed-in bytes.

This performs PKCS1 encryption of the passed-in data.

Parameters **bytes** (*bytes-like object*) – The value which will be encrypted.

Return type bytearray

Returns A PKCS1 encryption of the passed-in data.

static generate(*bits*, *key_type='rsa'*)
Generate a new key with the specified bit length.

Return type RSAKey

hasPrivateKey()
Return whether or not this key has a private component.

Return type bool

hashAndSign(*bytes*, *rsaScheme='PKCS1'*, *hAlg='sha1'*, *sLen=0*)
Hash and sign the passed-in bytes.

This requires the key to have a private component. It performs a PKCS1 or PSS signature on the passed-in data with selected hash algorithm.

Parameters

- **bytes** (*bytes-like object*) – The value which will be hashed and signed.
- **rsaScheme** (*str*) – The type of RSA scheme that will be applied, "PKCS1" for RSASSA-PKCS#1 v1.5 signature and "PSS" for RSASSA-PSS with MGF1 signature method
- **hAlg** (*str*) – The hash algorithm that will be used
- **sLen** (*int*) – The length of intended salt value, applicable only for RSASSA-PSS signatures

Return type bytearray

Returns A PKCS1 or PSS signature on the passed-in data.

hashAndVerify(*sigBytes*, *bytes*, *rsaScheme='PKCS1'*, *hAlg='sha1'*, *sLen=0*)
Hash and verify the passed-in bytes with the signature.

This verifies a PKCS1 or PSS signature on the passed-in data with selected hash algorithm.

Parameters

- **sigBytes** (*bytes-like object*) – A PKCS1 or PSS signature.
- **bytes** (*bytes-like object*) – The value which will be hashed and verified.
- **rsaScheme** (*str*) – The type of RSA scheme that will be applied, "PKCS1" for RSASSA-PKCS#1 v1.5 signature and "PSS" for RSASSA-PSS with MGF1 signature method
- **hAlg** (*str*) – The hash algorithm that will be used
- **sLen** (*int*) – The length of intended salt value, applicable only for RSASSA-PSS signatures

Return type bool

Returns Whether the signature matches the passed-in data.

sign(*bytes*, *padding='pkcs1'*, *hashAlg=None*, *saltLen=None*)
Sign the passed-in bytes.

This requires the key to have a private component. It performs a PKCS1 signature on the passed-in data.

Parameters

- **bytes** (*bytes-like object*) – The value which will be signed.

- **padding** (*str*) – name of the rsa padding mode to use, supported: "pkcs1" for RSASSA-PKCS1_1_5 and "pss" for RSASSA-PSS.
- **hashAlg** (*str*) – name of hash to be encoded using the PKCS#1 prefix for "pkcs1" padding or the hash used for MGF1 in "pss". Parameter is mandatory for "pss" padding.
- **saltLen** (*int*) – length of salt used for the PSS padding. Default is the length of the hash output used.

Return type bytearray

Returns A PKCS1 signature on the passed-in data.

verify(*sigBytes*, *bytes*, *padding='pkcs1'*, *hashAlg=None*, *saltLen=None*)
Verify the passed-in bytes with the signature.

This verifies a PKCS1 signature on the passed-in data.

Parameters

- **sigBytes** (*bytes-like object*) – A PKCS1 signature.
- **bytes** (*bytes-like object*) – The value which will be verified.

Return type bool

Returns Whether the signature matches the passed-in data.

write(*password=None*)
Return a string containing the key.

Return type str

Returns A string describing the key, in whichever format (PEM) is native to the implementation.

tlslite.utils.rsakey.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.utils.tackwrapper module

tlslite.utils.tlshashlib module

hashlib that handles FIPS mode.

tlslite.utils.tlshashlib.**md5**(**args*, ***kwargs*)
MD5 constructor that works in FIPS mode.

tlslite.utils.tlshashlib.**new**(**args*, ***kwargs*)
General constructor that works in FIPS mode.

tlslite.utils.tripledes module

Abstract class for 3DES.

class tlslite.utils.tripledes.**TripleDES**(*key*, *mode*, *IV*, *implementation*)
Bases: object

__init__(*key*, *mode*, *IV*, *implementation*)
Initialize self. See help(type(self)) for accurate signature.

decrypt(*ciphertext*)

encrypt(*plaintext*)

tlslite.utils.x25519 module

Handling X25519 and X448 curve based key agreement protocol.

tlslite.utils.x25519.**cswap**(*swap*, *x_2*, *x_3*)
Conditional swap function.

tlslite.utils.x25519.**decodeScalar22519**(*k*)
Function to decode the private K parameter of the x25519 function.

tlslite.utils.x25519.**decodeScalar448**(*k*)
Function to decode the private K parameter of the X448 function.

tlslite.utils.x25519.**decodeUCoordinate**(*u*, *bits*)
Function to decode the public U coordinate of X25519-family curves.

tlslite.utils.x25519.**x25519**(*k*, *u*)
Perform point multiplication on X25519 curve.

Parameters

- **k** (*bytearray*) – random secret value (multiplier), should be 32 byte long
- **u** (*bytearray*) – curve generator or the other party key share

Return type bytearray

tlslite.utils.x25519.**x448**(*k*, *u*)
Perform point multiplication on X448 curve.

Parameters

- **k** (*bytearray*) – random secret value (multiplier), should be 56 bytes long
- **u** (*bytearray*) – curve generator or the other party key share

Return type bytearray

1.1.2 Submodules

tlslite.api module

tlslite.basedb module

Base class for SharedKeyDB and VerifierDB.

class tlslite.basedb.**BaseDB**(*filename*, *type*)
Bases: object

__contains__(*username*)
Check if the database contains the specified username.

Parameters **username** (*str*) – The username to check for.

Return type bool

Returns True if the database contains the username, False otherwise.

__delitem__(*username*)

__getitem__(*username*)

__init__(*filename*, *type*)
Initialize self. See help(type(self)) for accurate signature.

__setitem__(*username*, *value*)

check(*username*, *param*)

create()
Create a new on-disk database.

Raises **anydbm.error** – If there's a problem creating the database.

keys()
Return a list of usernames in the database.

Return type list

Returns The usernames in the database.

open()
Open a pre-existing on-disk database.

Raises

- **anydbm.error** – If there's a problem opening the database.
- **ValueError** – If the database is not of the right type.

tlslite.bufferedsocket module

Wrapper around the socket.socket interface that provides buffering

class tlslite.bufferedsocket.**BufferedSocket**(*socket*)
Bases: object

Socket that will buffer reads and writes to a real socket object

When buffer_writes is enabled, writes won't be passed to the real socket until flush() is called.

Not multithread safe.

Variables **buffer_writes** (*boolean*) – whether to buffer data writes, False by default

__init__(*socket*)
Associate socket with the object

close()
Close the underlying socket.

flush()
Send all buffered data

getpeername()
Return the remote address to which the socket is connected

(socket emulation)

getsockname()
Return the socket's own address (socket emulation).

gettimeout()
Return the timeout associated with socket operations

(socket emulation)

recv(*bufsize*)
Receive data from socket (socket emulation)

send(*data*)
Send data to the socket

sendall(*data*)
Send data to the socket

setsockopt(*level*, *optname*, *value*)
Set the value of the given socket option (socket emulation).

settimeout(*value*)
Set a timeout on blocking socket operations (socket emulation).

shutdown(*how*)
Shutdown the underlying socket.

tlslite.checker module

Class for post-handshake certificate checking.

class tlslite.checker.**Checker**(*x509Fingerprint=None*, *checkResumedSession=False*)
Bases: object

This class is passed to a handshake function to check the other party's certificate chain.

If a handshake function completes successfully, but the Checker judges the other party's certificate chain to be missing or inadequate, a subclass of *tlslite.errors.TLSAuthenticationError* will be raised.

Currently, the Checker can check an X.509 chain.

__call__(*connection*)
Check a TLSConnection.

When a Checker is passed to a handshake function, this will be called at the end of the function.

Parameters **connection** (tlslite.tlsconnection.TLSConnection) – The TLSConnection to examine.

Raises ***tlslite.errors.TLSAuthenticationError*** – If the other party's certificate chain is missing or bad.

__init__(*x509Fingerprint=None*, *checkResumedSession=False*)
Create a new Checker instance.

You must pass in one of these argument combinations:

- x509Fingerprint

Parameters

- **x509Fingerprint** (*str*) – A hex-encoded X.509 end-entity fingerprint which the other party's end-entity certificate must match.
- **checkResumedSession** (*bool*) – If resumed sessions should be checked. This defaults to False, on the theory that if the session was checked once, we don't need to bother re-checking it.

tlslite.constants module

class tlslite.constants.**AlertDescription**
Bases: *tlslite.constants.TLSEnum*

Variables

- ***bad_record_mac*** – A TLS record failed to decrypt properly.

  If this occurs during a SRP handshake it most likely indicates a bad password. It may also indicate an implementation error, or some tampering with the data in transit.

  This alert will be signalled by the server if the SRP password is bad. It may also be signalled by the server if the SRP username is unknown to the server, but it doesn't wish to reveal that fact.

- ***handshake_failure*** – A problem occurred while handshaking.

  This typically indicates a lack of common ciphersuites between client and server, or some other disagreement (about SRP parameters or key sizes, for example).

- ***protocol_version*** – The other party's SSL/TLS version was unacceptable.

  This indicates that the client and server couldn't agree on which version of SSL or TLS to use.

- ***user_canceled*** – The handshake is being cancelled for some reason.

access_denied = 49

bad_certificate = 42

bad_certificate_hash_value = 114

bad_certificate_status_response = 113

bad_record_mac = 20

certificate_expired = 45

certificate_required = 116

certificate_revoked = 44

certificate_unknown = 46

certificate_unobtainable = 111

close_notify = 0

decode_error = 50

decompression_failure = 30

decrypt_error = 51

decryption_failed = 21

export_restriction = 60

handshake_failure = 40

illegal_parameter = 47

inappropriate_fallback = 86

insufficient_security = 71

internal_error = 80

missing_extension = 109

no_application_protocol = 120

no_certificate = 41

no_renegotiation = 100

protocol_version = 70

record_overflow = 22

unexpected_message = 10

unknown_ca = 48

unknown_psk_identity = 115

unrecognized_name = 112

unsupported_certificate = 43

unsupported_extension = 110

user_canceled = 90

class tlslite.constants.**AlertLevel**
Bases: *tlslite.constants.TLSEnum*

Enumeration of TLS Alert protocol levels

fatal = 2

warning = 1

class tlslite.constants.**AlgorithmOID**
Bases: *tlslite.constants.TLSEnum*

Algorithm OIDs as defined in rfc5758(ecdsa), rfc5754(rsa, sha), rfc3447(rss-pss). The key is the DER encoded OID in hex and the value is the algorithm id.

oid = {b'\x06\x07*\x86H\xce8\x04\x03': (2, 2), b'\x06\x07*\x86H\xce=\x04\x01': (2, 3

class tlslite.constants.**CertificateStatusType**
Bases: *tlslite.constants.TLSEnum*

Type of responses in the status_request and CertificateStatus msgs.

ocsp = 1

class tlslite.constants.**CertificateType**
Bases: *tlslite.constants.TLSEnum*

openpgp = 1

x509 = 0

class tlslite.constants.**CipherSuite**

Bases: object

Numeric values of ciphersuites and ciphersuite types

Variables

- ***tripleDESSuites*** – ciphersuties which use 3DES symmetric cipher in CBC mode
- ***aes128Suites*** – ciphersuites which use AES symmetric cipher in CBC mode with 128 bit key
- ***aes256Suites*** – ciphersuites which use AES symmetric cipher in CBC mode with 256 bit key
- ***rc4Suites*** – ciphersuites which use RC4 symmetric cipher with 128 bit key
- ***shaSuites*** – ciphersuites which use SHA-1 HMAC integrity mechanism and protocol default Pseudo Random Function
- ***sha256Suites*** – ciphersuites which use SHA-256 HMAC integrity mechanism and SHA-256 Pseudo Random Function
- ***md5Suites*** – ciphersuites which use MD-5 HMAC integrity mechanism and protocol default Pseudo Random Function
- ***srpSuites*** – ciphersuites which use Secure Remote Password (SRP) key exchange protocol
- ***srpCertSuites*** – ciphersuites which use Secure Remote Password (SRP) key exchange protocol with RSA server authentication
- ***srpAllSuites*** – all SRP ciphersuites, pure SRP and with RSA based server authentication
- ***certSuites*** – ciphersuites which use RSA key exchange with RSA server authentication
- ***certAllSuites*** – ciphersuites which use RSA server authentication
- ***anonSuites*** – ciphersuites which use anonymous Finite Field Diffie-Hellman key exchange
- ***ietfNames*** – dictionary with string names of the ciphersuites

SSL_CK_DES_192_EDE3_CBC_WITH_MD5 = 458944

SSL_CK_DES_64_CBC_WITH_MD5 = 393280

SSL_CK_IDEA_128_CBC_WITH_MD5 = 327808

SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5 = 262272

SSL_CK_RC2_128_CBC_WITH_MD5 = 196736

SSL_CK_RC4_128_EXPORT40_WITH_MD5 = 131200

SSL_CK_RC4_128_WITH_MD5 = 65664

TLS_AES_128_CCM_8_SHA256 = 4869

TLS_AES_128_CCM_SHA256 = 4868

TLS_AES_128_GCM_SHA256 = 4865

TLS_AES_256_GCM_SHA384 = 4866

TLS_CHACHA20_POLY1305_SHA256 = 4867

```
TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA = 19
TLS_DHE_DSS_WITH_AES_128_CBC_SHA = 50
TLS_DHE_DSS_WITH_AES_128_CBC_SHA256 = 64
TLS_DHE_DSS_WITH_AES_128_GCM_SHA256 = 162
TLS_DHE_DSS_WITH_AES_256_CBC_SHA = 56
TLS_DHE_DSS_WITH_AES_256_CBC_SHA256 = 106
TLS_DHE_DSS_WITH_AES_256_GCM_SHA384 = 163
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA = 22
TLS_DHE_RSA_WITH_AES_128_CBC_SHA = 51
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 = 103
TLS_DHE_RSA_WITH_AES_128_CCM = 49310
TLS_DHE_RSA_WITH_AES_128_CCM_8 = 49314
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 = 158
TLS_DHE_RSA_WITH_AES_256_CBC_SHA = 57
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 = 107
TLS_DHE_RSA_WITH_AES_256_CCM = 49311
TLS_DHE_RSA_WITH_AES_256_CCM_8 = 49315
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 = 159
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 = 52394
TLS_DHE_RSA_WITH_CHACHA20_POLY1305_draft_00 = 52387
TLS_DH_ANON_WITH_3DES_EDE_CBC_SHA = 27
TLS_DH_ANON_WITH_AES_128_CBC_SHA = 52
TLS_DH_ANON_WITH_AES_128_CBC_SHA256 = 108
TLS_DH_ANON_WITH_AES_128_GCM_SHA256 = 166
TLS_DH_ANON_WITH_AES_256_CBC_SHA = 58
TLS_DH_ANON_WITH_AES_256_CBC_SHA256 = 109
TLS_DH_ANON_WITH_AES_256_GCM_SHA384 = 167
TLS_DH_ANON_WITH_RC4_128_MD5 = 24
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA = 13
TLS_DH_DSS_WITH_AES_128_CBC_SHA = 48
TLS_DH_DSS_WITH_AES_128_CBC_SHA256 = 62
TLS_DH_DSS_WITH_AES_128_GCM_SHA256 = 164
TLS_DH_DSS_WITH_AES_256_CBC_SHA = 54
TLS_DH_DSS_WITH_AES_256_CBC_SHA256 = 104
TLS_DH_DSS_WITH_AES_256_GCM_SHA384 = 165
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA = 49160
```

```
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA = 49161
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 = 49187
TLS_ECDHE_ECDSA_WITH_AES_128_CCM = 49324
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 = 49326
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 = 49195
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA = 49162
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 = 49188
TLS_ECDHE_ECDSA_WITH_AES_256_CCM = 49325
TLS_ECDHE_ECDSA_WITH_AES_256_CCM_8 = 49327
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 = 49196
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 = 52393
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_draft_00 = 52386
TLS_ECDHE_ECDSA_WITH_NULL_SHA = 49158
TLS_ECDHE_ECDSA_WITH_RC4_128_SHA = 49159
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA = 49170
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA = 49171
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 = 49191
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 = 49199
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA = 49172
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 = 49192
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 = 49200
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 = 52392
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_draft_00 = 52385
TLS_ECDHE_RSA_WITH_NULL_SHA = 49168
TLS_ECDHE_RSA_WITH_RC4_128_SHA = 49169
TLS_ECDH_ANON_WITH_3DES_EDE_CBC_SHA = 49175
TLS_ECDH_ANON_WITH_AES_128_CBC_SHA = 49176
TLS_ECDH_ANON_WITH_AES_256_CBC_SHA = 49177
TLS_ECDH_ANON_WITH_NULL_SHA = 49173
TLS_ECDH_ANON_WITH_RC4_128_SHA = 49174
TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA = 49155
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA = 49156
TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 = 49189
TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 = 49197
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA = 49157
TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 = 49190
```

TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384 = 49198

TLS_ECDH_ECDSA_WITH_NULL_SHA = 49153

TLS_ECDH_ECDSA_WITH_RC4_128_SHA = 49154

TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA = 49165

TLS_ECDH_RSA_WITH_AES_128_CBC_SHA = 49166

TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 = 49193

TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 = 49201

TLS_ECDH_RSA_WITH_AES_256_CBC_SHA = 49167

TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 = 49194

TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384 = 49202

TLS_ECDH_RSA_WITH_NULL_SHA = 49163

TLS_ECDH_RSA_WITH_RC4_128_SHA = 49164

TLS_EMPTY_RENEGOTIATION_INFO_SCSV = 255

TLS_FALLBACK_SCSV = 22016

TLS_RSA_WITH_3DES_EDE_CBC_SHA = 10

TLS_RSA_WITH_AES_128_CBC_SHA = 47

TLS_RSA_WITH_AES_128_CBC_SHA256 = 60

TLS_RSA_WITH_AES_128_CCM = 49308

TLS_RSA_WITH_AES_128_CCM_8 = 49312

TLS_RSA_WITH_AES_128_GCM_SHA256 = 156

TLS_RSA_WITH_AES_256_CBC_SHA = 53

TLS_RSA_WITH_AES_256_CBC_SHA256 = 61

TLS_RSA_WITH_AES_256_CCM = 49309

TLS_RSA_WITH_AES_256_CCM_8 = 49313

TLS_RSA_WITH_AES_256_GCM_SHA384 = 157

TLS_RSA_WITH_NULL_MD5 = 1

TLS_RSA_WITH_NULL_SHA = 2

TLS_RSA_WITH_NULL_SHA256 = 59

TLS_RSA_WITH_RC4_128_MD5 = 4

TLS_RSA_WITH_RC4_128_SHA = 5

TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA = 49180

TLS_SRP_SHA_DSS_WITH_AES_128_CBC_SHA = 49183

TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA = 49186

TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA = 49179

TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA = 49182

TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA = 49185

TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA = 49178

TLS_SRP_SHA_WITH_AES_128_CBC_SHA = 49181

TLS_SRP_SHA_WITH_AES_256_CBC_SHA = 49184

aeadSuites = [156, 158, 166, 49195, 49197, 49201, 49199, 4865, 162, 164, 157, 159, 167
AEAD integrity, any PRF

aes128CcmSuites = [49308, 49310, 49324, 4868]
AES-128 CCM ciphers

aes128Ccm_8Suites = [49312, 49314, 49326, 4869]
AES-128 CCM_8 ciphers

aes128GcmSuites = [156, 158, 166, 49195, 49197, 49201, 49199, 4865, 162, 164]
AES-128 GCM ciphers

aes128Suites = [49181, 49182, 47, 51, 52, 60, 103, 108, 49187, 49161, 49189, 49156, 49
AES-128 CBC ciphers

aes256CcmSuites = [49309, 49311, 49325]

aes256Ccm_8Suites = [49313, 49315, 49327]
AES-256 CCM_8 ciphers

aes256GcmSuites = [157, 159, 167, 49196, 49198, 49202, 49200, 4866, 163, 165]
AES-256-GCM ciphers (implicit SHA384, see sha384PrfSuites)

aes256Suites = [49184, 49185, 53, 58, 57, 61, 107, 109, 49188, 49162, 49190, 49157, 49
AES-256 CBC ciphers

anonSuites = [167, 166, 109, 58, 108, 52, 27, 24]
anon FFDHE key exchange

static canonicalCipherName(*ciphersuite*)
Return the canonical name of the cipher whose number is provided.

static canonicalMacName(*ciphersuite*)
Return the canonical name of the MAC whose number is provided.

certAllSuites = [49185, 49182, 49179, 157, 156, 49309, 49308, 61, 60, 53, 47, 49313, 4
RSA authentication

certSuites = [157, 156, 49309, 49308, 61, 60, 53, 47, 49313, 49312, 10, 5, 4, 1, 2, 59
RSA key exchange, RSA authentication

chacha20Suites = [52392, 52393, 52394, 4867]
CHACHA20 cipher (implicit POLY1305 authenticator, SHA256 PRF)

chacha20draft00Suites = [52385, 52386, 52387]
CHACHA20 cipher, 00'th IETF draft (implicit POLY1305 authenticator)

dhAllSuites = [52394, 52387, 159, 158, 49311, 49310, 107, 103, 57, 51, 49315, 49314, 2

dheCertSuites = [52394, 52387, 159, 158, 49311, 49310, 107, 103, 57, 51, 49315, 49314,
FFDHE key exchange, RSA authentication

ecdhAllSuites = [52393, 52386, 49196, 49195, 49325, 49324, 49188, 49187, 49162, 49161,
all ciphersuites which use ephemeral ECDH key exchange

ecdhAnonSuites = [49177, 49176, 49175, 49174, 49173]
anon ECDHE key exchange

ecdheCertSuites = [52392, 52385, 49200, 49199, 49192, 49191, 49172, 49171, 49170, 4916
ECDHE key exchange, RSA authentication

ecdheEcdsaSuites = [52393, 52386, 49196, 49195, 49325, 49324, 49188, 49187, 49162, 491
ECDHE key exchange, ECDSA authentication

static filterForVersion(*suites*, *minVersion*, *maxVersion*)
Return a copy of suites without ciphers incompatible with version

static filter_for_certificate(*suites*, *cert_chain*)
Return a copy of suites without ciphers incompatible with the cert.

classmethod getAnonSuites(*settings*, *version=None*)
Provide anonymous DH ciphersuites matching settings

classmethod getCertSuites(*settings*, *version=None*)
Return ciphers with RSA authentication matching settings

classmethod getDheCertSuites(*settings*, *version=None*)
Provide authenticated DHE ciphersuites matching settings

classmethod getEcdhAnonSuites(*settings*, *version=None*)
Provide anonymous ECDH ciphersuites matching settings

classmethod getEcdheCertSuites(*settings*, *version=None*)
Provide authenticated ECDHE ciphersuites matching settings

classmethod getEcdsaSuites(*settings*, *version=None*)
Provide ECDSA authenticated ciphersuites matching settings

classmethod getSrpAllSuites(*settings*, *version=None*)
Return all SRP cipher suites matching settings

classmethod getSrpCertSuites(*settings*, *version=None*)
Return SRP cipher suites that use server certificates

classmethod getSrpSuites(*settings*, *version=None*)
Return SRP cipher suites matching settings

classmethod getTLS13Suites(*settings*, *version=None*)
Return cipher suites that are TLS 1.3 specific.

ietfNames = {1: 'TLS_RSA_WITH_NULL_MD5', 2: 'TLS_RSA_WITH_NULL_SHA', 4: 'TLS_RSA_WI'

md5Suites = [24, 4, 1]
MD-5 HMAC, protocol default PRF

nullSuites = [1, 2, 59, 49158, 49153, 49163, 49168, 49173]
no encryption

rc4Suites = [49169, 49159, 49154, 49164, 24, 5, 4, 49174]
RC4 128 stream cipher

sha256Suites = [60, 61, 103, 107, 59, 108, 109, 49187, 49189, 49193, 49191]
SHA-256 HMAC, SHA-256 PRF

sha384PrfSuites = [49188, 49190, 49194, 49192, 163, 165, 157, 159, 167, 49196, 49198,
TLS1.2 with SHA384 PRF

sha384Suites = [49188, 49190, 49194, 49192, 163, 165]
SHA-384 HMAC, SHA-384 PRF

shaSuites = [49178, 49181, 49184, 49179, 49182, 49185, 49180, 49183, 49186, 10, 47, 53
SHA-1 HMAC, protocol default PRF

srpAllSuites = [49184, 49181, 49178, 49185, 49182, 49179]
All that use SRP key exchange

srpCertSuites = [49185, 49182, 49179]
SRP key exchange, RSA authentication

srpSuites = [49184, 49181, 49178]
SRP key exchange, no certificate base authentication

ssl2_128Key = [65664, 131200, 196736, 262272, 327808]
SSL2 ciphersuties which use 128 bit key

ssl2_192Key = [458944]
SSL2 ciphersuites which use 192 bit key

ssl2_3des = [458944]
SSL2 ciphersuites which use 3DES symmetric cipher

ssl2_64Key = [393280]
SSL2 ciphersuites which use 64 bit key

ssl2des = [393280]
SSL2 ciphersuites which use (single) DES symmetric cipher

ssl2export = [131200, 262272]
SSL2 ciphersuites which encrypt only part (40 bits) of the key

ssl2idea = [327808]
SSL2 ciphersuites which use IDEA symmetric cipher

ssl2rc2 = [196736, 262272]
SSL2 ciphersuites which use RC2 symmetric cipher

ssl2rc4 = [65664, 131200]
SSL2 ciphersuites which use RC4 symmetric cipher

ssl3Suites = [49178, 49181, 49184, 49179, 49182, 49185, 49180, 49183, 49186, 10, 47, 5
SSL3, TLS1.0, TLS1.1 and TLS1.2 compatible ciphers

streamSuites = [49169, 49159, 49154, 49164, 24, 5, 4, 49174, 1, 2, 59, 49158, 49153, 4
stream cipher construction

tls12Suites = [60, 61, 103, 107, 59, 108, 109, 49187, 49189, 49193, 49191, 49188, 4919
TLS1.2 specific ciphersuites

tls13Suites = [4866, 4865, 4867, 4868, 4869]
TLS1.3 specific ciphersuites

tripleDESSuites = [49160, 49155, 49165, 49170, 49178, 49179, 10, 22, 27, 49175, 13, 19
3DES CBC ciphers

class tlslite.constants.**ClientCertificateType**
Bases: *tlslite.constants.TLSEnum*

dss_fixed_dh = 4

dss_sign = 2

ecdsa_fixed_ecdh = 66

ecdsa_sign = 64

rsa_fixed_dh = 3

rsa_fixed_ecdh = 65

rsa_sign = 1

class tlslite.constants.**ContentType**
Bases: *tlslite.constants.TLSEnum*

TLS record layer content types of payloads

alert = 21

all = (20, 21, 22, 23, 24)

application_data = 23

change_cipher_spec = 20

handshake = 22

heartbeat = 24

classmethod toRepr(*value*, *blacklist=None*)
Convert numeric type to name representation

class tlslite.constants.**ECCurveType**
Bases: *tlslite.constants.TLSEnum*

Types of ECC curves supported in TLS from RFC4492

explicit_char2 = 2

explicit_prime = 1

named_curve = 3

class tlslite.constants.**ECPointFormat**
Bases: *tlslite.constants.TLSEnum*

Names and ID's of supported EC point formats.

all = [0, 1, 2]

ansiX962_compressed_char2 = 2

ansiX962_compressed_prime = 1

classmethod toRepr(*value*, *blacklist=None*)
Convert numeric type to name representation.

uncompressed = 0

class tlslite.constants.**ExtensionType**
Bases: *tlslite.constants.TLSEnum*

TLS Extension Type registry values

alpn = 16

cert_type = 9

client_hello_padding = 21

cookie = 44

early_data = 42

ec_point_formats = 11

encrypt_then_mac = 22

extended_master_secret = 23

```
    extended_random = 40
    heartbeat = 15
    key_share = 51
    post_handshake_auth = 49
    pre_shared_key = 41
    psk_key_exchange_modes = 45
    record_size_limit = 28
    renegotiation_info = 65281
    server_name = 0
    signature_algorithms = 13
    signature_algorithms_cert = 50
    srp = 12
    status_request = 5
    supported_groups = 10
    supported_versions = 43
    supports_npn = 13172
    tack = 62208
class tlslite.constants.Fault
    Bases: object
    badA = 103
    badB = 201
    badFinished = 300
    badMAC = 301
    badPadding = 302
    badPassword = 102
    badPremasterPadding = 501
    badUsername = 101
    badVerifyMessage = 601
    clientCertFaults = [601]
    clientNoAuthFaults = [501, 502]
    clientSrpFaults = [101, 102, 103]
    faultAlerts = {101:  (115, 20), 102:  (20,), 103:  (47,), 300:  (51,), 301:  (20,), 30
    faultNames = {101:  'bad username', 102:  'bad password', 103:  'bad A', 300:  'bad fi
    genericFaults = [300, 301, 302]
    serverFaults = [201]
    shortPremasterSecret = 502
```

class tlslite.constants.**GroupName**

Bases: *tlslite.constants.TLSEnum*

Name of groups supported for (EC)DH key exchange

all = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22,

allEC = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22

allFF = [256, 257, 258, 259, 260]

brainpoolP256r1 = 26

brainpoolP384r1 = 27

brainpoolP512r1 = 28

ffdhe2048 = 256

ffdhe3072 = 257

ffdhe4096 = 258

ffdhe6144 = 259

ffdhe8192 = 260

secp160k1 = 15

secp160r1 = 16

secp160r2 = 17

secp192k1 = 18

secp192r1 = 19

secp224k1 = 20

secp224r1 = 21

secp256k1 = 22

secp256r1 = 23

secp384r1 = 24

secp521r1 = 25

sect163k1 = 1

sect163r1 = 2

sect163r2 = 3

sect193r1 = 4

sect193r2 = 5

sect233k1 = 6

sect233r1 = 7

sect239k1 = 8

sect283k1 = 9

sect283r1 = 10

sect409k1 = 11

sect409r1 = 12

sect571k1 = 13

sect571r1 = 14

classmethod toRepr(*value*, *blacklist=None*)
Convert numeric type to name representation

x25519 = 29

x448 = 30

class tlslite.constants.**HandshakeType**
Bases: *tlslite.constants.TLSEnum*

Message types in TLS Handshake protocol

certificate = 11

certificate_request = 13

certificate_status = 22

certificate_verify = 15

client_hello = 1

client_key_exchange = 16

encrypted_extensions = 8

finished = 20

hello_request = 0

hello_retry_request = 6

key_update = 24

message_hash = 254

new_session_ticket = 4

next_protocol = 67

server_hello = 2

server_hello_done = 14

server_key_exchange = 12

class tlslite.constants.**HashAlgorithm**
Bases: *tlslite.constants.TLSEnum*

Hash algorithm IDs used in TLSv1.2

intrinsic = 8

md5 = 1

none = 0

sha1 = 2

sha224 = 3

sha256 = 4

sha384 = 5

sha512 = 6

class tlslite.constants.**HeartbeatMessageType**
Bases: *tlslite.constants.TLSEnum*

Types of heartbeat messages from RFC 6520

heartbeat_request = 1

heartbeat_response = 2

class tlslite.constants.**HeartbeatMode**
Bases: *tlslite.constants.TLSEnum*

Types of heartbeat modes from RFC 6520

PEER_ALLOWED_TO_SEND = 1

PEER_NOT_ALLOWED_TO_SEND = 2

class tlslite.constants.**KeyUpdateMessageType**
Bases: *tlslite.constants.TLSEnum*

Types of keyupdate messages from RFC 8446

update_not_requested = 0

update_requested = 1

class tlslite.constants.**NameType**
Bases: *tlslite.constants.TLSEnum*

Type of entries in Server Name Indication extension.

host_name = 0

class tlslite.constants.**PskKeyExchangeMode**
Bases: *tlslite.constants.TLSEnum*

Values used in the PSK Key Exchange Modes extension.

psk_dhe_ke = 1

psk_ke = 0

class tlslite.constants.**SSL2ErrorDescription**
Bases: *tlslite.constants.TLSEnum*

SSL2 Handshake protocol error message descriptions

bad_certificate = 4

no_certificate = 2

no_cipher = 1

unsupported_certificate_type = 6

class tlslite.constants.**SSL2HandshakeType**
Bases: *tlslite.constants.TLSEnum*

SSL2 Handshake Protocol message types.

client_certificate = 8

client_finished = 3

client_hello = 1

client_master_key = 2

error = 0

request_certificate = 7

server_finished = 6

server_hello = 4

server_verify = 5

class tlslite.constants.**SignatureAlgorithm**
Bases: *tlslite.constants.TLSEnum*

Signing algorithms used in TLSv1.2

anonymous = 0

dsa = 2

ecdsa = 3

ed25519 = 7

ed448 = 8

rsa = 1

class tlslite.constants.**SignatureScheme**
Bases: *tlslite.constants.TLSEnum*

Signature scheme used for signalling supported signature algorithms.

This is the replacement for the HashAlgorithm and SignatureAlgorithm lists. Introduced with TLSv1.3.

dsa_sha1 = (2, 2)

dsa_sha224 = (3, 2)

dsa_sha256 = (4, 2)

dsa_sha384 = (5, 2)

dsa_sha512 = (6, 2)

ecdsa_secp256r1_sha256 = (4, 3)

ecdsa_secp384r1_sha384 = (5, 3)

ecdsa_secp521r1_sha512 = (6, 3)

ecdsa_sha1 = (2, 3)

ecdsa_sha224 = (3, 3)

ed25519 = (8, 7)

ed448 = (8, 8)

static getHash(*scheme*)
Return the name of hash used in signature scheme.

static getKeyType(*scheme*)
Return the name of the signature algorithm used in scheme.

E.g. for "rsa_pkcs1_sha1" it returns "rsa"

static getPadding(*scheme*)
Return the name of padding scheme used in signature scheme.

rsa_pkcs1_sha1 = (2, 1)

rsa_pkcs1_sha224 = (3, 1)

rsa_pkcs1_sha256 = (4, 1)

rsa_pkcs1_sha384 = (5, 1)

rsa_pkcs1_sha512 = (6, 1)

rsa_pss_pss_sha256 = (8, 9)

rsa_pss_pss_sha384 = (8, 10)

rsa_pss_pss_sha512 = (8, 11)

rsa_pss_rsae_sha256 = (8, 4)

rsa_pss_rsae_sha384 = (8, 5)

rsa_pss_rsae_sha512 = (8, 6)

rsa_pss_sha256 = (8, 4)

rsa_pss_sha384 = (8, 5)

rsa_pss_sha512 = (8, 6)

classmethod toRepr(*value*, *blacklist=None*)
Convert numeric type to name representation

class tlslite.constants.**TLSEnum**
Bases: object

Base class for different enums of TLS IDs

classmethod toRepr(*value*, *blacklist=None*)
Convert numeric type to string representation

name if found, None otherwise

classmethod toStr(*value*, *blacklist=None*)
Convert numeric type to human-readable string if possible

tlslite.defragmenter module

Helper package for handling fragmentation of messages.

class tlslite.defragmenter.**Defragmenter**
Bases: object

Class for demultiplexing TLS messages.

Since the messages can be interleaved and fragmented between each other we need to cache not complete ones and return in order of urgency.

Supports messages with given size (like Alerts) or with a length header in specific place (like Handshake messages).

Variables

- **priorities** – order in which messages from given types should be returned.
- **buffers** – data buffers for message types
- **decoders** – functions which check buffers if a message of given type is complete

__init__()
Set up empty defregmenter

add_data(*msg_type*, *data*)
Adds data to buffers

add_dynamic_size(*msg_type*, *size_offset*, *size_of_size*)
Add a message type which has a dynamic size set in a header

add_static_size(*msg_type*, *size*)
Add a message type which all messages are of same length

clear_buffers()
Remove all data from buffers

get_message()
Extract the highest priority complete message from buffer

tlslite.dh module

Handling of Diffie-Hellman parameter files.

tlslite.dh.**parse**(*data*)
Parses DH parameters from a binary string.

The string can either by PEM or DER encoded

Parameters **data** (*bytes*) – DH parameters

Return type tuple of int

Returns generator and prime

tlslite.dh.**parseBinary**(*data*)
Parse DH parameters from ASN.1 DER encoded binary string.

Parameters **data** (*bytes*) – DH parameters

Return type tuple of int

tlslite.errors module

Exception classes.

exception tlslite.errors.**BaseTLSException**
Bases: Exception

Metaclass for TLS Lite exceptions.

Look to *tlslite.errors.TLSError* for exceptions that should be caught by tlslite consumers

exception tlslite.errors.**EncodingError**
Bases: *tlslite.errors.EncryptionError*

An error appeared while encoding

exception tlslite.errors.**EncryptionError**
Bases: *tlslite.errors.BaseTLSException*

Base class for exceptions thrown while encrypting.

exception tlslite.errors.**InvalidSignature**
Bases: *tlslite.errors.EncryptionError*

Verification function found invalid signature

exception tlslite.errors.**MaskTooLongError**
Bases: *tlslite.errors.EncryptionError*

The maskLen passed into function is too high

exception tlslite.errors.**MessageTooLongError**
Bases: *tlslite.errors.EncryptionError*

The message passed into function is too long

exception tlslite.errors.**TLSAbruptCloseError**
Bases: *tlslite.errors.TLSError*

The socket was closed without a proper TLS shutdown.

The TLS specification mandates that an alert of some sort must be sent before the underlying socket is closed. If the socket is closed without this, it could signify that an attacker is trying to truncate the connection. It could also signify a misbehaving TLS implementation, or a random network failure.

exception tlslite.errors.**TLSAlert**
Bases: *tlslite.errors.TLSError*

A TLS alert has been signalled.

exception tlslite.errors.**TLSAuthenticationError**
Bases: *tlslite.errors.TLSError*

The handshake succeeded, but the other party's authentication was inadequate.

This exception will only be raised when a `tlslite.Checker.Checker` has been passed to a handshake function. The Checker will be invoked once the handshake completes, and if the Checker objects to how the other party authenticated, a subclass of this exception will be raised.

exception tlslite.errors.**TLSAuthenticationTypeError**
Bases: *tlslite.errors.TLSAuthenticationError*

The Checker was expecting the other party to authenticate with a different type of certificate chain.

exception tlslite.errors.**TLSAuthorizationError**
Bases: *tlslite.errors.TLSAuthenticationError*

The Checker was expecting the other party to authenticate with a certificate chain that has a different authorization.

exception tlslite.errors.**TLSBadRecordMAC**
Bases: *tlslite.errors.TLSProtocolException*

Bad MAC (or padding in case of mac-then-encrypt)

exception tlslite.errors.**TLSClosedConnectionError**
Bases: *tlslite.errors.TLSError*, OSError

An attempt was made to use the connection after it was closed.

exception tlslite.errors.**TLSDecodeError**
Bases: *tlslite.errors.TLSProtocolException*

The received message encoding does not match specification.

exception tlslite.errors.**TLSDecryptionFailed**
Bases: *tlslite.errors.TLSProtocolException*

Decryption of data was unsuccessful

exception tlslite.errors.**TLSError**
Bases: *tlslite.errors.BaseTLSException*

Base class for all TLS Lite exceptions.

__str__()
At least print out the Exception time for str(...).

exception tlslite.errors.**TLSFaultError**
Bases: *tlslite.errors.TLSError*

The other party responded incorrectly to an induced fault.

This exception will only occur during fault testing, when a *tlslite.tlsconnection.TLSConnection*'s fault variable is set to induce some sort of faulty behavior, and the other party doesn't respond appropriately.

exception tlslite.errors.**TLSFingerprintError**
Bases: *tlslite.errors.TLSAuthenticationError*

The Checker was expecting the other party to authenticate with a certificate chain that matches a different fingerprint.

exception tlslite.errors.**TLSHandshakeFailure**
Bases: *tlslite.errors.TLSProtocolException*

Could not find acceptable set of handshake parameters

exception tlslite.errors.**TLSIllegalParameterException**
Bases: *tlslite.errors.TLSProtocolException*

Parameters specified in message were incorrect or invalid

exception tlslite.errors.**TLSInsufficientSecurity**
Bases: *tlslite.errors.TLSProtocolException*

Parameters selected by user are too weak

exception tlslite.errors.**TLSInternalError**
Bases: *tlslite.errors.TLSError*

The internal state of object is unexpected or invalid.

Caused by incorrect use of API.

exception tlslite.errors.**TLSLocalAlert**(*alert*, *message=None*)
Bases: *tlslite.errors.TLSAlert*

A TLS alert has been signalled by the local implementation.

Variables

- **description** (*int*) – Set to one of the constants in *tlslite.constants.AlertDescription*
- **level** (*int*) – Set to one of the constants in *tlslite.constants.AlertLevel*
- **message** (*str*) – Description of what went wrong.

__init__(*alert*, *message=None*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
At least print out the Exception time for str(...).

exception tlslite.errors.**TLSNoAuthenticationError**
Bases: *tlslite.errors.TLSAuthenticationError*

The Checker was expecting the other party to authenticate with a certificate chain, but this did not occur.

exception tlslite.errors.**TLSProtocolException**
Bases: *tlslite.errors.BaseTLSException*

Exceptions used internally for handling errors in received messages

exception tlslite.errors.**TLSRecordOverflow**
Bases: *tlslite.errors.TLSProtocolException*

The received record size was too big

exception tlslite.errors.**TLSRemoteAlert**(*alert*)
Bases: *tlslite.errors.TLSAlert*

A TLS alert has been signalled by the remote implementation.

Variables

- **description** (*int*) – Set to one of the constants in *tlslite.constants.AlertDescription*
- **level** (*int*) – Set to one of the constants in *tlslite.constants.AlertLevel*

__init__(*alert*)
Initialize self. See help(type(self)) for accurate signature.

__str__()
At least print out the Exception time for str(...).

exception tlslite.errors.**TLSUnexpectedMessage**
Bases: *tlslite.errors.TLSProtocolException*

The received message was unexpected or parsing of Inner Plaintext failed

exception tlslite.errors.**TLSUnknownPSKIdentity**
Bases: *tlslite.errors.TLSProtocolException*

The PSK or SRP identity is unknown

exception tlslite.errors.**TLSUnsupportedError**
Bases: *tlslite.errors.TLSError*

The implementation doesn't support the requested (or required) capabilities.

exception tlslite.errors.**TLSValidationError**(*msg*, *info=None*)
Bases: *tlslite.errors.TLSAuthenticationError*

The Checker has determined that the other party's certificate chain is invalid.

__init__(*msg*, *info=None*)
Initialize self. See help(type(self)) for accurate signature.

exception tlslite.errors.**UnknownRSAType**
Bases: *tlslite.errors.EncryptionError*

Unknown RSA algorithm type passed

tlslite.extensions module

Helper package for handling TLS extensions encountered in ClientHello and ServerHello messages.

class tlslite.extensions.**ALPNExtension**
Bases: *tlslite.extensions.TLSExtension*

Handling of Application Layer Protocol Negotiation extension from RFC 7301.

Variables

- **protocol_names** (*list of bytearrays*) – list of protocol names acceptable or selected by peer
- **extType** (*int*) – numberic type of ALPNExtension, i.e. 16
- **extData** (*bytearray*) – raw encoding of the extension data

__init__()
Create instance of ALPNExtension

See also: *create()* and *parse()*

__repr__()
Create programmer-readable representation of object

Return type str

create(*protocol_names=None*)
Create an instance of ALPNExtension with specified protocols

Parameters **protocols** (*list*) – list of protocol names that are to be sent

extData
Return encoded payload of the extension

Return type bytearray

parse(*parser*)
Parse the extension from on the wire format

Parameters **parser** (Parser) – data to be parsed as extension

Raises ***DecodeError*** – when the encoding of the extension is self inconsistent

Return type *ALPNExtension*

class tlslite.extensions.**CertificateStatusExtension**
Bases: *tlslite.extensions.TLSExtension*

Handling of Certificate Status response as redefined in TLS1.3

__init__()
Create instance of CertificateStatusExtension.

create(*status_type*, *response*)
Set values of the extension.

extData
Serialise the object.

parse(*parser*)
Deserialise the data from on the wire representation.

class tlslite.extensions.**ClientCertTypeExtension**
Bases: *tlslite.extensions.VarListExtension*

This class handles the (client variant of) Certificate Type extension

See RFC 6091.

Variables

- **extType** (*int*) – numeric type of Certificate Type extension, i.e. 9
- **extData** (*bytearray*) – raw representation of the extension data
- **certTypes** (*list of int*) – list of certificate type identifiers (each one byte long)

__init__()
Create an instance of ClientCertTypeExtension

See also: `create()` and `parse()`

class tlslite.extensions.**ClientKeyShareExtension**
Bases: *tlslite.extensions.TLSExtension*

Class for handling the Client Hello version of the Key Share extension.

Extension for sending the key shares to server

__init__()
Create instance of the object.

create(*client_shares*)
Set the advertised client shares in the extension.

extData
Return the on the wire raw encoding of the extension

Return type bytearray

parse(*parser*)
Parse the extension from on the wire format

Parameters **parser** (`Parser`) – data to be parsed

Raises ***DecodeError*** – when the data does not match the definition

Return type *ClientKeyShareExtension*

class tlslite.extensions.**CookieExtension**
Bases: *tlslite.extensions.VarBytesExtension*

Handling of the TLS 1.3 cookie extension.

__init__()
Create instance.

class tlslite.extensions.**CustomNameExtension**(*field_name*, *extType*)
Bases: *tlslite.extensions.TLSExtension*

Abstract class for handling custom name for payload variable.

Used for handling arbitrary extensions that deal with a single value in payload (either an opaque data, single value or single array).

Must be subclassed.

__init__(*field_name*, *extType*)
Create instance of the class.

Parameters

- **field_name** (*str*) – name of the field to store the extension payload
- **ext_type** (*int*) – numerical ID of the extension

create(*values*)
Set the list to specified values.

Parameters **values** (*list*) – list of values to save

extData
Return raw data encoding of the extension.

Return type bytearray

parse(*parser*)
Deserialise extension from on-the-wire data.

Parameters **parser** (tlslite.utils.codec.Parser) – data

Return type Extension

class tlslite.extensions.**ECPointFormatsExtension**
Bases: *tlslite.extensions.VarListExtension*

Client side list of supported ECC point formats.

See RFC4492.

Variables **formats** (*list of int*) – list of point formats supported by peer

__init__()
Create instance of class

class tlslite.extensions.**HRRKeyShareExtension**
Bases: *tlslite.extensions.TLSExtension*

Class for handling the Hello Retry Request variant of the Key Share ext.

Extension for notifying the client of the server selected group for key exchange.

__init__()
Create instance of the object.

create(*selected_group*)
Set the selected group in the extension.

extData
Serialise the payload of the extension.

parse(*parser*)
Parse the extension from on the wire format.

Parameters **parser** (Parser) – data to be parsed

Return type *HRRKeyShareExtension*

class tlslite.extensions.**HeartbeatExtension**
Bases: *tlslite.extensions.IntExtension*

Heartbeat extension from RFC 6520

Variables **mode** – mode if peer is allowed or nor allowed to send responses

__init__()
Create handler for extension that has a single integer as payload.

Parameters

- **field_name** (*str*) – name of the field that will store the value
- **elem_length** (*int*) – size (in bytes) of the value in the extension
- **ext_tyoe** (*int*) – numerical ID of the extension encoded
- **item_enum** (*class*) – TLSEnum class that defines entries in the extension

parse(*parser*)
Deserialise the extension from on the wire data.

class tlslite.extensions.**IntExtension**(*elem_length*, *field_name*, *ext_type*, *item_enum=None*)
Bases: *tlslite.extensions.CustomNameExtension*

Abstract class for extensions that deal with single integer in payload.

Extension for handling arbitrary extensions that have a payload that is a single integer of a given size (1, 2, ... bytes)

__init__(*elem_length*, *field_name*, *ext_type*, *item_enum=None*)
Create handler for extension that has a single integer as payload.

Parameters

- **field_name** (*str*) – name of the field that will store the value
- **elem_length** (*int*) – size (in bytes) of the value in the extension
- **ext_tyoe** (*int*) – numerical ID of the extension encoded
- **item_enum** (*class*) – TLSEnum class that defines entries in the extension

__repr__()
Return human readable representation of the extension.

extData
Return raw data encoding of the extension.

Return type bytearray

parse(*parser*)
Deserialise extension from on-the-wire data.

Parameters **parser** (tlslite.utils.codec.Parser) – data

Return type Extension

class tlslite.extensions.**KeyShareEntry**
Bases: object

Handler for of the item of the Key Share extension.

__init__()
Initialise the object.

create(*group*, *key_exchange*, *private=None*)
Initialise the Key Share Entry from Key Share extension.

Parameters

- **group** (*int*) – ID of the key share
- **key_exchange** (*bytearray*) – value of the key share

- **private** (*object*) – private value for the given share (won't be encoded during serialisation)

Return type *KeyShareEntry*

parse(*parser*)
Parse the value from on the wire format.

Parameters **parser** (Parser) – data to be parsed as extension

Return type *KeyShareEntry*

write(*writer*)
Write the on the wire representation of the item to writer.

Parameters **writer** (Writer) – buffer to write the data to

class tlslite.extensions.**ListExtension**(*fieldName*, *extType*, *item_enum=None*)
Bases: *tlslite.extensions.CustomNameExtension*

Abstract class for extensions that deal with single list in payload.

Extension for handling arbitrary extensions comprising of just a list of same-sized elementes inside an array

__init__(*fieldName*, *extType*, *item_enum=None*)
Create instance of the class.

Parameters

- **fieldName** (*str*) – name of the field to store the list that is the payload
- **extType** (*int*) – numerical ID of the extension
- **item_enum** (*class*) – TLSEnum class that defines the enum of the items in the list

__repr__()
Return human readable representation of the extension.

class tlslite.extensions.**NPNExtension**
Bases: *tlslite.extensions.TLSExtension*

This class handles the unofficial Next Protocol Negotiation TLS extension.

Variables

- **protocols** (*list of bytearrays*) – list of protocol names supported by the server
- **extType** (*int*) – numeric type of NPNExtension, i.e. 13172
- **extData** (*bytearray*) – raw representation of extension data

__init__()
Create an instance of NPNExtension

See also: *create()* and *parse()*

__repr__()
Create programmer-readable version of representation

Return type str

create(*protocols=None*)
Create an instance of NPNExtension with specified protocols

Parameters **protocols** (*list*) – list of protocol names that are supported

extData
Return the raw data encoding of the extension

Return type bytearray

parse(*p*)
Parse the extension from on the wire format

Parameters **p** (Parser) – data to be parsed

Raises ***DecodeError*** – when the size of the passed element doesn't match the internal representation

Return type *NPNExtension*

class tlslite.extensions.**PaddingExtension**
Bases: *tlslite.extensions.TLSExtension*

ClientHello message padding with a desired size.

Can be used to pad ClientHello messages to a desired size in order to avoid implementation bugs caused by certain ClientHello sizes.

See RFC7685.

__init__()
Create instance of class.

create(*size*)
Set the padding size and create null byte padding of defined size.

Parameters **size** (*int*) – required padding size in bytes

extData
Return raw encoding of the extension.

Return type bytearray

parse(*p*)
Deserialise extension from on the wire data.

Parameters **p** (Parser) – data to be parsed

Raises ***DecodeError*** – when the size of the passed element doesn't match the internal representation

Return type *TLSExtension*

class tlslite.extensions.**PreSharedKeyExtension**
Bases: *tlslite.extensions.TLSExtension*

Class for handling Pre Shared Key negotiation.

__init__()
Create instance of class.

create(*identities*, *binders*)
Set list of offered PSKs.

extData
Serialise the payload of the extension.

parse(*parser*)
Parse the extension from on the wire format.

class tlslite.extensions.**PskIdentity**
Bases: object

Handling of PskIdentity from PreSharedKey Extension.

__init__()
Create instance of class.

create(*identity*, *obf_ticket_age*)
Initialise instance.

parse(*parser*)
Deserialize the object from bytearray.

write()
Serialise the object.

class tlslite.extensions.**PskKeyExchangeModesExtension**
Bases: *tlslite.extensions.VarListExtension*

Handling of the PSK Key Exchange Modes extension.

__init__()
Create instance of class.

class tlslite.extensions.**RecordSizeLimitExtension**
Bases: *tlslite.extensions.IntExtension*

Class for handling the record_size_limit extension from RFC 8449.

__init__()
Create instance.

class tlslite.extensions.**RenegotiationInfoExtension**
Bases: *tlslite.extensions.VarBytesExtension*

Client and Server Hello secure renegotiation extension from RFC 5746

Should have an empty renegotiated_connection field in case of initial connection

__init__()
Create instance

class tlslite.extensions.**SNIExtension**
Bases: *tlslite.extensions.TLSExtension*

Class for handling Server Name Indication (server_name) extension from RFC 4366.

Note that while usually the client does advertise just one name, it is possible to provide a list of names, each of different type. The type is a single byte value (represented by ints), the names are opaque byte strings, in case of DNS host names (records of type 0) they are UTF-8 encoded domain names (without the ending dot).

Variables

- ***hostNames*** (*tuple of bytearrays*) – tuple of hostnames (server name records of type 0) advertised in the extension. Note that it may not include all names from client hello as the client can advertise other types. Also note that while it's not possible to change the returned array in place, it is possible to assign a new set of names. IOW, this won't work:

```
sni_extension.hostNames[0] = bytearray(b'example.com')
```

while this will work:

```
names = list(sni_extension.hostNames)
names[0] = bytearray(b'example.com')
sni_extension.hostNames = names
```

- **serverNames** (list of *ServerName*) – list of all names advertised in extension. *ServerName* is a namedtuple with two elements, the first element (type) defines the type of the name (encoded as int) while the other (name) is a bytearray that carries the value. Known types are defined in *tlslite.constants.NameType*. The list will be empty if the on the wire extension had and empty list while it will be None if the extension was empty.
- **extType** (*int*) – numeric type of SNIExtension, i.e. 0
- **extData** (*bytearray*) – raw representation of the extension

class ServerName(*name_type*, *name*)

Bases: tuple

__repr__()

Return a nicely formatted representation string

name

Alias for field number 1

name_type

Alias for field number 0

__init__()

Create an instance of SNIExtension.

See also: *create()* and *parse()*.

__repr__()

Return programmer-readable representation of extension

Return type str

create(*hostname=None*, *hostNames=None*, *serverNames=None*)

Initializes an instance with provided hostname, host names or raw server names.

Any of the parameters may be *None*, in that case the list inside the extension won't be defined, if either *hostNames* or *serverNames* is an empty list, then the extension will define a list of length 0.

If multiple parameters are specified at the same time, then the resulting list of names will be concatenated in order of hostname, hostNames and serverNames last.

Parameters

- **hostname** (*bytearray*) – raw UTF-8 encoding of the host name
- **hostNames** (*list*) – list of raw UTF-8 encoded host names
- **serverNames** (*list*) – pairs of name_type and name encoded as a namedtuple

Return type *SNIExtension*

extData

Raw encoding of extension data, without type and length header.

Return type bytearray

hostNames

Returns a simulated list of hostNames from the extension.

Return type tuple of bytearrays

parse(*p*)

Deserialise the extension from on-the-wire data

The parser should not include the type or length of extension!

Parameters **p** (*tlslite.util.codec.Parser*) – data to be parsed

Return type *SNIExtension*

Raises ***DecodeError*** – when the internal sizes don't match the attached data

write()
Returns encoded extension, as encoded on the wire

Return type bytearray

Returns an array of bytes formatted as they are supposed to be written on the wire, including the type, length and extension data

class tlslite.extensions.**SRPExtension**
Bases: *tlslite.extensions.TLSExtension*

This class handles the Secure Remote Password protocol TLS extension defined in RFC 5054.

Variables

- **extType** (*int*) – numeric type of SRPExtension, i.e. 12
- **extData** (*bytearray*) – raw representation of extension data
- **identity** (*bytearray*) – UTF-8 encoding of user name

__init__()
Create an instance of SRPExtension

See also: *create()* and *parse()*

__repr__()
Return programmer-centric description of extension

Return type str

create(*identity=None*)
Create and instance of SRPExtension with specified protocols

Parameters **identity** (*bytearray*) – UTF-8 encoded identity (user name) to be provided to user. MUST be shorter than 2^8-1.

Raises **ValueError** – when the identity lenght is longer than 2^8-1

extData
Return raw data encoding of the extension

Return type bytearray

parse(*p*)
Parse the extension from on the wire format

Parameters **p** (Parser) – data to be parsed

Raises ***DecodeError*** – when the data is internally inconsistent

Return type *SRPExtension*

class tlslite.extensions.**ServerCertTypeExtension**
Bases: *tlslite.extensions.IntExtension*

This class handles the Certificate Type extension (variant sent by server) defined in RFC 6091.

Variables

- **extType** (*int*) – binary type of Certificate Type extension, i.e. 9

- **extData** (*bytearray*) – raw representation of the extension data
- ***cert_type*** (*int*) – the certificate type selected by server

__init__()
Create an instance of ServerCertTypeExtension

See also: `create()` and *parse()*

parse(*parser*)
Parse the extension from on the wire format

Parameters **p** (`Parser`) – parser with data

class tlslite.extensions.**ServerKeyShareExtension**
Bases: *tlslite.extensions.TLSExtension*

Class for handling the Server Hello variant of the Key Share extension.

Extension for sending the key shares to client

__init__()
Create instance of the object.

create(*server_share*)
Set the advertised server share in the extension.

extData
Serialise the payload of the extension

parse(*parser*)
Parse the extension from on the wire format.

Parameters **parser** (`Parser`) – data to be parsed

Return type *ServerKeyShareExtension*

class tlslite.extensions.**SignatureAlgorithmsCertExtension**
Bases: tlslite.extensions._SigListExt

Client side list of supported signatures in certificates.

Should be used when the signatures supported in certificates and in TLS messages differ.

See TLS1.3 RFC

__init__()
Create instance of class.

class tlslite.extensions.**SignatureAlgorithmsExtension**
Bases: tlslite.extensions._SigListExt

Client side list of supported signature algorithms.

Should be used by server to select certificate and signing method for Server Key Exchange messages. In practice used only for the latter.

See RFC5246.

__init__()
Create instance of class

class tlslite.extensions.**SrvPreSharedKeyExtension**
Bases: *tlslite.extensions.IntExtension*

Handling of the Pre Shared Key extension from server.

__init__()
Create instance of class.

class tlslite.extensions.**SrvSupportedVersionsExtension**
Bases: *tlslite.extensions.TLSExtension*

Handling of SupportedVersion extension in SH and HRR in TLS 1.3.

See draft-ietf-tls-tls13.

Variables

- **extType** (*int*) – numeric type of the Supported Versions extension, i.e. 43
- **extData** (*bytearray*) – raw representation of the extension payload
- **version** (*tuple*) – version selected by the server

__init__()
Creates a generic TLS extension.

You'll need to use *create()* or *parse()* methods to create an extension that is actually usable.

Parameters

- **server** (*bool*) – whether to select ClientHello or ServerHello version for parsing
- **extType** (*int*) – type of extension encoded as an integer, to be used by subclasses
- **encExt** (*bool*) – whether to select the EncryptedExtensions type for parsing
- **cert** (*bool*) – whether to select the Certificate type of extension for parsing
- **hrr** (*bool*) – whether to select the Hello Retry Request type of extension for parsing

__repr__()
Return programmer-readable representation of the extension.

Return type str

create(*version*)
Set the version supported by the server.

Parameters **version** (*tuple*) – Version selected by server.

Return type *SrvSupportedVersionsExtension*

extData
Raw encoding of extension data, without type and length header.

Return type bytearray

parse(*parser*)
Deserialise the extension from on-the-wire data.

The parser should not include the type or length of extension.

Parameters **parser** (*tlslite.util.codec.Parser*) – data to be parsed

Return type *SrvSupportedVersionsExtension*

class tlslite.extensions.**StatusRequestExtension**
Bases: *tlslite.extensions.TLSExtension*

Handling of the Certificate Status Request extension from RFC 6066.

Variables

- **status_type** (*int*) – type of the status request

- **responder_id_list** (*list of bytearray*) – list of DER encoded OCSP responder identifiers that the client trusts
- **request_extensions** (*bytearray*) – DER encoded list of OCSP extensions, as defined in RFC 2560

__init__()
Create instance of StatusRequestExtension.

__repr__()
Create programmer-readable representation of object

Return type str

create(*status_type=1*, *responder_id_list=()*, *request_extensions=b''*)
Create an instance of StatusRequestExtension with specified options.

Parameters

- **status_type** (*int*) – type of status returned
- **responder_id_list** (*list*) – list of encoded OCSP responder identifiers that the client trusts
- **request_extensions** (*bytearray*) – DER encoding of requested OCSP extensions

extData
Return encoded payload of the extension.

Return type bytearray

parse(*parser*)
Parse the extension from on the wire format.

Parameters **parser** (Parser) – data to be parsed as extension

Return type *StatusRequestExtension*

class tlslite.extensions.**SupportedGroupsExtension**
Bases: *tlslite.extensions.VarListExtension*

Client side list of supported groups of (EC)DHE key exchage.

See RFC4492, RFC7027 and RFC-ietf-tls-negotiated-ff-dhe-10

Variables **groups** (*int*) – list of groups that the client supports

__init__()
Create instance of class

class tlslite.extensions.**SupportedVersionsExtension**
Bases: *tlslite.extensions.VarSeqListExtension*

This class handles the SupportedVersion extensions used in TLS 1.3.

See draft-ietf-tls-tls13.

Variables

- **extType** (*int*) – numeric type of the Supported Versions extension, i.e. 43
- **extData** (*bytearray*) – raw representation of the extension data
- **versions** (*list of tuples*) – list of supported protocol versions; each tuple has two one byte long integers

__init__()
Create an instance of SupportedVersionsExtension.

class tlslite.extensions.**TACKExtension**
Bases: *tlslite.extensions.TLSExtension*

This class handles the server side TACK extension (see draft-perrin-tls-tack-02).

Variables

- **tacks** (*list*) – list of TACK's supported by server
- **activation_flags** (*int*) – activation flags for the tacks

class TACK
Bases: object

Implementation of the single TACK

__eq__(*other*)
Tests if the other object is equivalent to this TACK

Returns False for every object that's not a TACK

__init__()
Create a single TACK object

__repr__()
Return programmmer readable representation of TACK object
Return type str

create(*public_key*, *min_generation*, *generation*, *expiration*, *target_hash*, *signature*)
Initialise the TACK with data

parse(*p*)
Parse the TACK from on the wire format
Parameters **p** (Parser) – data to be parsed
Return type *TACK*
Raises ***DecodeError*** – when the internal sizes don't match the provided data

write()
Convert the TACK into on the wire format
Return type bytearray

__init__()
Create an instance of TACKExtension

See also: *create()* and :py:meth'parse'

__repr__()
Create a programmer readable representation of TACK extension

Return type str

create(*tacks*, *activation_flags*)
Initialize the instance of TACKExtension

Return type *TACKExtension*

extData
Return the raw data encoding of the extension

Return type bytearray

parse(*p*)
Parse the extension from on the wire format

Parameters **p** (Parser) – data to be parsed

Return type *TACKExtension*

class tlslite.extensions.**TLSExtension**(*server=False*, *extType=None*, *encExt=False*, *cert=False*, *hrr=False*)
Bases: object

Base class for handling handshake protocol hello messages extensions.

This class handles the generic information about TLS extensions used by both sides of connection in Client Hello and Server Hello messages. See https://tools.ietf.org/html/rfc4366 for more info.

It is used as a base class for specific users and as a way to store extensions that are not implemented in library.

To implement a new extension you will need to create a new class which calls this class contructor (__init__), usually specifying just the extType parameter. The other methods which need to be implemented are: *extData*, *create*, *parse* and *__repr__*. If the parser can be used for client and optionally server extensions, the extension constructor should be added to *_universalExtensions*. Otherwise, when the client and server extensions have completely different forms, you should add client form to the *_universalExtensions* and the server form to *_serverExtensions*. Since the server MUST NOT send extensions not advertised by client, there are no purely server-side extensions. But if the client side extension is just marked by presence and has no payload, the client side (thus the *_universalExtensions* may be skipped, then the *TLSExtension* class will be used for implementing it. See end of the file for type-to-constructor bindings.

Note: Subclassing for the purpose of parsing extensions is not an officially supported part of API (just as underscores in their names would indicate).

Variables

- **extType** (*int*) – a 2^16-1 limited integer specifying the type of the extension that it contains, e.g. 0 indicates server name extension
- **extData** (*bytearray*) – a byte array containing the value of the extension as to be written on the wire
- **serverType** (*boolean*) – indicates that the extension was parsed with ServerHello specific parser, otherwise it used universal or ClientHello specific parser
- **encExtType** (*boolean*) – indicates that the extension should be the type from Encrypted Extensions
- **_universalExtensions** (*dict*) – dictionary with concrete implementations of specific TLS extensions where key is the numeric value of the extension ID. Contains ClientHello version of extensions or universal implementations
- **_serverExtensions** (*dict*) – dictionary with concrete implementations of specific TLS extensions where key is the numeric value of the extension ID. Includes only those extensions that require special handlers for ServerHello versions.
- **_certificateExtensions** (*dict*) – dictionary with concrete implementations of specific TLS extensions where the key is the numeric value of the type of the extension and the value is the class. Includes only those extensions that require special handlers for Certificate message.
- **_hrrExtensions** (*dict*) – dictionary with concrete implementation of specific TLS extensions where the key is the numeric type of the extension and the value is the class.

Includes only those extensions that require special handlers for the Hello Retry Request message.

__eq__(*that*)
Test if two TLS extensions are effectively the same

Will check if encoding them will result in the same on the wire representation.

Will return False for every object that's not an extension.

__init__(*server=False*, *extType=None*, *encExt=False*, *cert=False*, *hrr=False*)
Creates a generic TLS extension.

You'll need to use `create()` or `parse()` methods to create an extension that is actually usable.

Parameters

- **server** (*bool*) – whether to select ClientHello or ServerHello version for parsing
- **extType** (*int*) – type of extension encoded as an integer, to be used by subclasses
- **encExt** (*bool*) – whether to select the EncryptedExtensions type for parsing
- **cert** (*bool*) – whether to select the Certificate type of extension for parsing
- **hrr** (*bool*) – whether to select the Hello Retry Request type of extension for parsing

__repr__()
Output human readable representation of object

Child classes should override this method to support more appropriate string rendering of the extension.

Return type str

create(**args*, ***kwargs*)
Initializes a generic TLS extension.

The extension can carry arbitrary data and have arbitrary payload, can be used in client hello or server hello messages.

The legacy calling method uses two arguments - the *extType* and *data*. If the new calling method is used, only one argument is passed in - *data*.

Child classes need to override this method so that it is possible to set values for all fields used by the extension.

Parameters

- **extType** (*int*) – if int: type of the extension encoded as an integer between *0* and *2^16-1*
- **data** (*bytearray*) – raw data representing extension on the wire

Return type *TLSExtension*

extData
Return the on the wire encoding of extension

Child classes need to override this property so that it returns just the payload of an extension, that is, without the 4 byte generic header common to all extension. In other words, without the extension ID and overall extension length.

Return type bytearray

parse(*p*)
Parses extension from on the wire format

Child classes should override this method so that it parses the extension from on the wire data. Note that child class parsers will not receive the generic header of the extension, but just a parser with the payload. In other words, the method should be the exact reverse of the *extData* property.

Parameters **p** (*tlslite.util.codec.Parser*) – data to be parsed

Raises **DecodeError** – when the size of the passed element doesn't match the internal representation

Return type *TLSExtension*

write()

Returns encoded extension, as encoded on the wire

Note that child classes in general don't need to override this method.

Return type bytearray

Returns An array of bytes formatted as is supposed to be written on the wire, including the extension_type, length and the extension data

Raises **AssertionError** – when the object was not initialized

class tlslite.extensions.**VarBytesExtension**(*field_name*, *length_length*, *ext_type*)

Bases: *tlslite.extensions.CustomNameExtension*

Abstract class for extension that deal with single byte array payload.

Extension for handling arbitrary extensions that comprise of just a single bytearray of variable size.

__init__(*field_name*, *length_length*, *ext_type*)

Crate instance of the class.

Parameters

- **field_name** (*str*) – name of the field to store the bytearray that is the payload
- **length_length** (*int*) – number of bytes needed to encode the length field of the string
- **ext_type** (*int*) – numerical ID of the extension

__repr__()

Return human readable representation of the extension.

extData

Return raw data encoding of the extension.

Return type bytearray

parse(*parser*)

Deserialise extension from on-the-wire data.

Parameters **parser** (tlslite.utils.codec.Parser) – data

Return type *TLSExtension*

class tlslite.extensions.**VarListExtension**(*elemLength*, *lengthLength*, *fieldName*, *extType*, *item_enum=None*)

Bases: *tlslite.extensions.ListExtension*

Abstract extension for handling extensions comprised of uniform value list.

Extension for handling arbitrary extensions comprising of just a list of same-sized elementes inside an array

__init__(*elemLength*, *lengthLength*, *fieldName*, *extType*, *item_enum=None*)

Create handler for extension that has a list of items as payload.

Parameters

- **elemLength** (*int*) – number of bytes needed to encode single element
- **lengthLength** (*int*) – number of bytes needed to encode length field
- **fieldName** (*str*) – name of the field storing the list of elements
- **extType** (*int*) – numerical ID of the extension encoded
- **item_enum** (*class*) – TLSEnum class that defines entries in the list

extData
Return raw data encoding of the extension.

Return type bytearray

parse(*parser*)
Deserialise extension from on-the-wire data.

Parameters **parser** (tlslite.utils.codec.Parser) – data

Return type Extension

class tlslite.extensions.**VarSeqListExtension**(*elemLength*, *elemNum*, *lengthLength*, *fieldName*, *extType*, *item_enum=None*)
Bases: *tlslite.extensions.ListExtension*

Abstract extension for handling extensions comprised of tuple list.

Extension for handling arbitrary extensions comprising of a single list of same-sized elements in same-sized tuples

__init__(*elemLength*, *elemNum*, *lengthLength*, *fieldName*, *extType*, *item_enum=None*)
Create a handler for extension that has a list of tuples as payload.

Parameters

- **elemLength** (*int*) – number of bytes needed to encode single element of a tuple
- **elemNum** (*int*) – number of elements in a tuple
- **lengthLength** (*int*) – number of bytes needed to encode overall length of the list
- **fieldName** (*str*) – name of the field storing the list of elements
- **extType** (*int*) – numerical ID of the extension encoded
- **item_enum** (*class*) – TLSEnum class that defines entries in the list

extData
Return raw data encoding of the extension.

Return type bytearray

parse(*parser*)
Deserialise extension from on-the-wire data.

Parameters **parser** (tlslite.utils.codec.Parser) – data

Return type Extension

tlslite.handshakehashes module

Handling cryptographic hashes for handshake protocol

class tlslite.handshakehashes.**HandshakeHashes**
Bases: object

Store and calculate necessary hashes for handshake protocol

Calculates message digests of messages exchanged in handshake protocol of SSLv3 and TLS.

__init__()
Create instance

copy()
Copy object

Return a copy of the object with all the hashes in the same state as the source object.

Return type *HandshakeHashes*

digest(*digest=None*)
Calculate and return digest for the already consumed data.

Used for Finished and CertificateVerify messages.

Parameters **digest** (*str*) – name of digest to return

digestSSL(*masterSecret*, *label*)
Calculate and return digest for already consumed data (SSLv3 version)

Used for Finished and CertificateVerify messages.

Parameters

- **masterSecret** (*bytearray*) – value of the master secret
- **label** (*bytearray*) – label to include in the calculation

update(*data*)
Add *data* to hash input.

Parameters **data** (*bytearray*) – serialized TLS handshake message

tlslite.handshakehelpers module

Class with various handshake helpers.

class tlslite.handshakehelpers.**HandshakeHelpers**
Bases: object

This class encapsulates helper functions to be used with a TLS handshake.

static alignClientHelloPadding(*clientHello*)
Align ClientHello using the Padding extension to 512 bytes at least.

Parameters **clientHello** (ClientHello) – ClientHello to be aligned

static calc_res_binder_psk(*iden*, *res_master_secret*, *tickets*)
Calculate PSK associated with provided ticket identity.

static update_binders(*client_hello*, *handshake_hashes*, *psk_configs*, *tickets=None*, *res_master_secret=None*)
Sign the Client Hello using TLS 1.3 PSK binders.

note: the psk_configs should be in the same order as the ones in the PreSharedKeyExtension extension (extra ones are ok)

Parameters

- **client_hello** – ClientHello to sign
- **handshake_hashes** – hashes of messages exchanged so far
- **psk_configs** – PSK identities and secrets
- **tickets** – optional list of tickets received from server
- **res_master_secret** (*bytearray*) – secret associated with the tickets

static verify_binder(*client_hello*, *handshake_hashes*, *position*, *secret*, *prf*, *external=True*)
Verify the PSK binder value in client hello.

Parameters

- **client_hello** – ClientHello to verify
- **handshake_hashes** – hashes of messages exchanged so far
- **position** – binder at which position should be verified
- **secret** – the secret PSK
- **prf** – name of the hash used as PRF

tlslite.handshakesettings module

Class for setting handshake parameters.

class tlslite.handshakesettings.**HandshakeSettings**
Bases: object

This class encapsulates various parameters that can be used with a TLS handshake.

Variables

- **minKeySize** (*int*) – The minimum bit length for asymmetric keys.

  If the other party tries to use SRP, RSA, or Diffie-Hellman parameters smaller than this length, an alert will be signalled. The default is 1023.

- **maxKeySize** (*int*) – The maximum bit length for asymmetric keys.

  If the other party tries to use SRP, RSA, or Diffie-Hellman parameters larger than this length, an alert will be signalled. The default is 8193.

- **cipherNames** (*list(str)*) – The allowed ciphers.

  The allowed values in this list are 'chacha20-poly1305', 'aes256gcm', 'aes128gcm', 'aes256', 'aes128', '3des', 'chacha20-poly1305_draft00', 'null' and 'rc4'. If these settings are used with a client handshake, they determine the order of the ciphersuites offered in the ClientHello message.

  If these settings are used with a server handshake, the server will choose whichever ciphersuite matches the earliest entry in this list.

  The default value is list that excludes 'rc4', 'null' and 'chacha20-poly1305_draft00'.

- **macNames** (*list(str)*) – The allowed MAC algorithms.

  The allowed values in this list are 'sha384', 'sha256', 'aead', 'sha' and 'md5'.

  The default value is list that excludes 'md5'.

- **certificateTypes** (*list(str)*) – The allowed certificate types.

  The only allowed certificate type is 'x509'. This list is only used with a client handshake. The client will advertise to the server which certificate types are supported, and will check that the server uses one of the appropriate types.

- **minVersion** (*tuple*) – The minimum allowed SSL/TLS version.

  This variable can be set to (3, 0) for SSL 3.0, (3, 1) for TLS 1.0, (3, 2) for TLS 1.1, or (3, 3) for TLS 1.2. If the other party wishes to use a lower version, a protocol_version alert will be signalled. The default is (3, 1).

- **maxVersion** (*tuple*) – The maximum allowed SSL/TLS version.

  This variable can be set to (3, 0) for SSL 3.0, (3, 1) for TLS 1.0, (3, 2) for TLS 1.1, or (3, 3) for TLS 1.2. If the other party wishes to use a higher version, a protocol_version alert will be signalled. The default is (3, 3).

  Warning: Some servers may (improperly) reject clients which offer support for TLS 1.1 or higher. In this case, try lowering maxVersion to (3, 1).

- **useExperimentalTackExtension** (*bool*) – Whether to enabled TACK support.

  Note that TACK support is not standardized by IETF and uses a temporary TLS Extension number, so should NOT be used in production software.

- **sendFallbackSCSV** (*bool*) – Whether to, as a client, send FALLBACK_SCSV.

- **rsaSigHashes** (*list(str)*) – List of hashes supported (and advertised as such) for TLS 1.2 signatures over Server Key Exchange or Certificate Verify with RSA signature algorithm.

  The list is sorted from most wanted to least wanted algorithm.

  The allowed hashes are: "md5", "sha1", "sha224", "sha256", "sha384" and "sha512". The default list does not include md5.

- **ecdsaSigHashes** (*list(str)*) – List of hashes supported (and advertised as such) for TLS 1.2 signatures over Server Key Exchange or Certificate Verify with ECDSA signature algorithm.

  The list is sorted from most wanted to least wanted algorithm.

  The allowed hashes are: "sha1", "sha224", "sha256", "sha384" and "sha512".

- **eccCurves** (*list(str)*) – List of named curves that are to be supported

- **useEncryptThenMAC** (*bool*) – whether to support the encrypt then MAC extension from RFC 7366. True by default.

- **useExtendedMasterSecret** (*bool*) – whether to support the extended master secret calculation from RFC 7627. True by default.

- **requireExtendedMasterSecret** (*bool*) – whether to require negotiation of extended master secret calculation for successful connection. Requires useExtendedMasterSecret to be set to true. False by default.

- **defaultCurve** (*str*) – curve that will be used by server in case the client did not advertise support for any curves. It does not have to be the first curve for eccCurves and may be distinct from curves from that list.

- **keyShares** (*list(str)*) – list of TLS 1.3 key shares to include in Client Hello

- **padding_cb** (*func*) – Callback to function computing number of padding bytes for TLS 1.3. Signature is cb_func(msg_size, content_type, max_size).
- **pskConfigs** (*list(tuple(bytearray, bytearray, bytearray))*) – list of tuples, first element of the tuple is the human readable, UTF-8 encoded, "identity" of the associated secret (bytearray, can be empty for TLS 1.2 and earlier), second element is the binary secret (bytearray), third is an optional parameter specifying the PRF hash to be used in TLS 1.3 (`sha256` or `sha384`)
- **ticketKeys** (*list(bytearray)*) – keys to be used for encrypting and decrypting session tickets. First entry is the encryption key for new tickets and the default decryption key, subsequent entries are the fallback keys allowing for key rollover. The keys need to be of size appropriate for a selected cipher in ticketCipher, 32 bytes for 'aes256gcm' and 'chacha20-poly1305', 16 bytes for 'aes128-gcm'. New keys should be generated regularly and replace old ones. Key use time should generally not be longer than 24h and key life-time should not be longer than 48h. Leave empty to disable session ticket support on server side.
- **ticketCipher** (*str*) – name of the cipher used for encrypting the session tickets. 'aes256gcm' by default, 'aes128gcm' or 'chacha20-poly1305' alternatively.
- **ticketLifetime** (*int*) – maximum allowed lifetime of ticket encryption key, in seconds. 1 day by default
- **psk_modes** (*list(str)*) – acceptable modes for the PSK key exchange in TLS 1.3
- **max_early_data** (*int*) – maximum number of bytes acceptable for 0-RTT early_data processing. In other words, how many bytes will the server try to process, but ignore, in case the Client Hello includes early_data extension.
- **use_heartbeat_extension** (*bool*) – whether to support heartbeat extension from RFC 6520. True by default.
- **heartbeat_response_callback** (*func*) – Callback to function when Heartbeat response is received.
- **record_size_limit** (*int*) – maximum size of records we are willing to process (value advertised to the other side). It must not be larger than 2**14+1 (the maximum for TLS 1.3) and will be reduced to 2**14 if TLS 1.2 or lower is the highest enabled version. Must not be set to values smaller than 64. Set to None to disable support for the extension. See also: RFC 8449.
- **keyExchangeNames** (*list*) – Enabled key exchange types for the connection, influences selected cipher suites.

__init__()
Initialise default values for settings.

getCertificateTypes()
Get list of certificate types as IDs

validate()
Validate the settings, filter out unsupported ciphersuites and return a copy of object. Does not modify the original object.

Return type *HandshakeSettings*

Returns a self-consistent copy of settings

Raises **ValueError** – when settings are invalid, insecure or unsupported.

class tlslite.handshakesettings.**Keypair**(*key=None*, *certificates=()*)
Bases: object

Key, certificate and related data.

Stores also certificate associate data like OCSPs and transparency info. TODO: add the above

First certificate in certificates needs to match key, remaining should build a trust path to a root CA.

Variables

- **key** (RSAKey *or* ECDSAKey) – private key
- **certificates** (*list* *(X509)*) – the certificates to send to peer if the key is selected for use. The first one MUST include the public key of the `key`

__init__(*key=None*, *certificates=()*)
Initialize self. See help(type(self)) for accurate signature.

validate()
Sanity check the keypair.

class tlslite.handshakesettings.**VirtualHost**
Bases: object

Configuration of keys and certs for a single virual server.

This class encapsulates keys and certificates for hosts specified by server_name (SNI) and ALPN extensions.

TODO: support SRP as alternative to certificates TODO: support PSK as alternative to certificates

Variables

- ***keys*** (*list* *(Keypair)*) – List of certificates and keys to be used in this virtual host. First keypair able to server ClientHello will be used.
- **hostnames** (*set* *(bytes)*) – all the hostnames that server supports please use *matches_hostname()* to verify if the VirtualHost can serve a request to a given hostname as that allows wildcard hosts that always reply True.
- **trust_anchors** (*list* *(X509)*) – list of CA certificates supported for client certificate authentication, sent in CertificateRequest
- **app_protocols** (*list* *(bytes)*) – all the application protocols that the server supports (for ALPN)

__init__()
Set up default configuration.

matches_hostname(*hostname*)
Checks if the virtual host can serve hostname

validate()
Sanity check the settings

tlslite.keyexchange module

Handling of cryptographic operations for key exchange

class tlslite.keyexchange.**ADHKeyExchange**(*cipherSuite*, *clientHello*, *serverHello*, *dhParams=None*, *dhGroups=None*)
Bases: *tlslite.keyexchange.KeyExchange*

Handling of anonymous Diffie-Hellman Key exchange

FFDHE without signing serverKeyExchange useful for anonymous DH

__init__(*cipherSuite*, *clientHello*, *serverHello*, *dhParams=None*, *dhGroups=None*)
Initialize KeyExchange. privateKey is the signing private key

makeClientKeyExchange()
Create client key share for the key exchange

makeServerKeyExchange()
Prepare server side of anonymous key exchange with selected parameters

processClientKeyExchange(*clientKeyExchange*)
Use client provided parameters to establish premaster secret

processServerKeyExchange(*srvPublicKey*, *serverKeyExchange*)
Process the server key exchange, return premaster secret.

class tlslite.keyexchange.**AECDHKeyExchange**(*cipherSuite*, *clientHello*, *serverHello*, *acceptedCurves*, *defaultCurve=23*)
Bases: *tlslite.keyexchange.KeyExchange*

Handling of anonymous Eliptic curve Diffie-Hellman Key exchange

ECDHE without signing serverKeyExchange useful for anonymous ECDH

__init__(*cipherSuite*, *clientHello*, *serverHello*, *acceptedCurves*, *defaultCurve=23*)
Initialize KeyExchange. privateKey is the signing private key

makeClientKeyExchange()
Make client key exchange for ECDHE

makeServerKeyExchange(*sigHash=None*)
Create AECDHE version of Server Key Exchange

processClientKeyExchange(*clientKeyExchange*)
Calculate premaster secret from previously generated SKE and CKE

processServerKeyExchange(*srvPublicKey*, *serverKeyExchange*)
Process the server key exchange, return premaster secret

class tlslite.keyexchange.**AuthenticatedKeyExchange**(*cipherSuite*, *clientHello*, *serverHello*, *privateKey=None*)
Bases: *tlslite.keyexchange.KeyExchange*

Common methods for key exchanges that authenticate Server Key Exchange

Methods for signing Server Key Exchange message

makeServerKeyExchange(*sigHash=None*)
Prepare server side of key exchange with selected parameters

class tlslite.keyexchange.**DHE_RSAKeyExchange**(*cipherSuite*, *clientHello*, *serverHello*, *privateKey*, *dhParams=None*, *dhGroups=None*)
Bases: *tlslite.keyexchange.AuthenticatedKeyExchange*, *tlslite.keyexchange.ADHKeyExchange*

Handling of authenticated ephemeral Diffe-Hellman Key exchange.

__init__(*cipherSuite*, *clientHello*, *serverHello*, *privateKey*, *dhParams=None*, *dhGroups=None*)
Create helper object for Diffie-Hellamn key exchange.

Parameters **dhParams** (*2-element tuple of int*) – Diffie-Hellman parameters that will be used by server. First element of the tuple is the generator, the second is the prime. If not specified it will use a secure set (currently a 2048-bit safe prime).

class tlslite.keyexchange.**ECDHE_RSAKeyExchange**(*cipherSuite*, *clientHello*, *serverHello*, *privateKey*, *acceptedCurves*, *defaultCurve=23*)

Bases: *tlslite.keyexchange.AuthenticatedKeyExchange*, *tlslite.keyexchange.AECDHKeyExchange*

Helper class for conducting ECDHE key exchange

__init__(*cipherSuite*, *clientHello*, *serverHello*, *privateKey*, *acceptedCurves*, *defaultCurve=23*)
Initialize KeyExchange. privateKey is the signing private key

class tlslite.keyexchange.**ECDHKeyExchange**(*group*, *version*)

Bases: *tlslite.keyexchange.RawDHKeyExchange*

Implementation of the Elliptic Curve Diffie-Hellman key exchange.

__init__(*group*, *version*)
Set the parameters of the key exchange

Sets group on which the KEX will take part and protocol version used.

calc_public_value(*private*)
Calculate public value for given private key.

calc_shared_key(*private*, *peer_share*)
Calculate the shared key,

get_random_private_key()
Return random private key value for the selected curve.

class tlslite.keyexchange.**FFDHKeyExchange**(*group*, *version*, *generator=None*, *prime=None*)

Bases: *tlslite.keyexchange.RawDHKeyExchange*

Implemenation of the Finite Field Diffie-Hellman key exchange.

__init__(*group*, *version*, *generator=None*, *prime=None*)
Set the parameters of the key exchange

Sets group on which the KEX will take part and protocol version used.

calc_public_value(*private*)
Calculate the public value for given private value.

Return type int

calc_shared_key(*private*, *peer_share*)
Calculate the shared key.

get_random_private_key()
Return a random private value for the prime used.

Return type int

class tlslite.keyexchange.**KeyExchange**(*cipherSuite*, *clientHello*, *serverHello*, *privateKey=None*)

Bases: object

Common API for calculating Premaster secret

NOT stable, will get moved from this file

__init__(*cipherSuite*, *clientHello*, *serverHello*, *privateKey=None*)
Initialize KeyExchange. privateKey is the signing private key

static calcVerifyBytes(*version*, *handshakeHashes*, *signatureAlg*, *premasterSecret*, *clientRandom*, *serverRandom*, *prf_name=None*, *peer_tag=b'client'*, *key_type='rsa'*)
Calculate signed bytes for Certificate Verify

static makeCertificateVerify(*version*, *handshakeHashes*, *validSigAlgs*, *privateKey*, *certificateRequest*, *premasterSecret*, *clientRandom*, *serverRandom*)
Create a Certificate Verify message

Parameters

- **version** – protocol version in use
- **handshakeHashes** – the running hash of all handshake messages
- **validSigAlgs** – acceptable signature algorithms for client side, applicable only to TLSv1.2 (or later)
- **certificateRequest** – the server provided Certificate Request message
- **premasterSecret** – the premaster secret, needed only for SSLv3
- **clientRandom** – client provided random value, needed only for SSLv3
- **serverRandom** – server provided random value, needed only for SSLv3

makeClientKeyExchange()
Create a ClientKeyExchange object

Returns a ClientKeyExchange for the second flight from client in the handshake.

makeServerKeyExchange(*sigHash=None*)
Create a ServerKeyExchange object

Returns a ServerKeyExchange object for the server's initial leg in the handshake. If the key exchange method does not send ServerKeyExchange (e.g. RSA), it returns None.

processClientKeyExchange(*clientKeyExchange*)
Process ClientKeyExchange and return premaster secret

Processes the client's ClientKeyExchange message and returns the premaster secret. Raises TLSLocalAlert on error.

processServerKeyExchange(*srvPublicKey*, *serverKeyExchange*)
Process the server KEX and return premaster secret

signServerKeyExchange(*serverKeyExchange*, *sigHash=None*)
Sign a server key exchange using default or specified algorithm

Parameters **sigHash** (*str*) – name of the signature hash to be used for signing

static verifyServerKeyExchange(*serverKeyExchange*, *publicKey*, *clientRandom*, *serverRandom*, *validSigAlgs*)
Verify signature on the Server Key Exchange message

the only acceptable signature algorithms are specified by validSigAlgs

class tlslite.keyexchange.**RSAKeyExchange**(*cipherSuite*, *clientHello*, *serverHello*, *privateKey*)
Bases: *tlslite.keyexchange.KeyExchange*

Handling of RSA key exchange

NOT stable API, do NOT use

__init__(*cipherSuite*, *clientHello*, *serverHello*, *privateKey*)
Initialize KeyExchange. privateKey is the signing private key

makeClientKeyExchange()
Return a client key exchange with clients key share

makeServerKeyExchange(*sigHash=None*)
Don't create a server key exchange for RSA key exchange

processClientKeyExchange(*clientKeyExchange*)
Decrypt client key exchange, return premaster secret

processServerKeyExchange(*srvPublicKey*, *serverKeyExchange*)
Generate premaster secret for server

class tlslite.keyexchange.**RawDHKeyExchange**(*group*, *version*)
Bases: object

Abstract class for performing Diffe-Hellman key exchange.

Provides a shared API for X25519, ECDHE and FFDHE key exchange.

__init__(*group*, *version*)
Set the parameters of the key exchange

Sets group on which the KEX will take part and protocol version used.

calc_public_value(*private*)
Calculate the public value from the provided private value.

calc_shared_key(*private*, *peer_share*)
Calcualte the shared key given our private and remote share value

get_random_private_key()
Generate a random value suitable for use as the private value of KEX.

class tlslite.keyexchange.**SRPKeyExchange**(*cipherSuite*, *clientHello*, *serverHello*, *privateKey*, *verifierDB*, *srpUsername=None*, *password=None*, *settings=None*)
Bases: *tlslite.keyexchange.KeyExchange*

Helper class for conducting SRP key exchange

__init__(*cipherSuite*, *clientHello*, *serverHello*, *privateKey*, *verifierDB*, *srpUsername=None*, *password=None*, *settings=None*)
Link Key Exchange options with verifierDB for SRP

makeClientKeyExchange()
Create ClientKeyExchange

makeServerKeyExchange(*sigHash=None*)
Create SRP version of Server Key Exchange

processClientKeyExchange(*clientKeyExchange*)
Calculate premaster secret from Client Key Exchange and sent SKE

processServerKeyExchange(*srvPublicKey*, *serverKeyExchange*)
Calculate premaster secret from ServerKeyExchange

tlslite.mathtls module

Miscellaneous helper functions.

tlslite.mathtls.**FFDHE_PARAMETERS = {'RFC2409 group 1': (2, 155251809230070893513091813125**
Listing of all well known FFDH parameters.

Please note that this dictionary includes all groups that are well-known (i.e. named), irrespective if their use is recommended or not.

You should use RFC7919_GROUPS for well-known secure groups.

class tlslite.mathtls.**MAC_SSL**
Bases: object

copy()

create(*k*, *digestmod=None*)

digest()

update(*m*)

tlslite.mathtls.**PAD**(*n*, *x*)

tlslite.mathtls.**PRF**(*secret*, *label*, *seed*, *length*)

tlslite.mathtls.**PRF_1_2**(*secret*, *label*, *seed*, *length*)
Pseudo Random Function for TLS1.2 ciphers that use SHA256

tlslite.mathtls.**PRF_1_2_SHA384**(*secret*, *label*, *seed*, *length*)
Pseudo Random Function for TLS1.2 ciphers that use SHA384

tlslite.mathtls.**PRF_SSL**(*secret*, *seed*, *length*)

tlslite.mathtls.**P_hash**(*mac_name*, *secret*, *seed*, *length*)
Internal method for calculation the PRF in TLS.

tlslite.mathtls.**RFC7919_GROUPS = [(2, 323170060713110073001535134778251633624880571334890750**
All DH parameters specified in RFC 7919.

Those are the parameters recommended for use in TLS.

tlslite.mathtls.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.mathtls.**calcExtendedMasterSecret**(*version*, *cipherSuite*, *premasterSecret*, *handshakeHashes*)
Derive Extended Master Secret from premaster and handshake msgs

tlslite.mathtls.**calcFinished**(*version*, *masterSecret*, *cipherSuite*, *handshakeHashes*, *isClient*)
Calculate the Handshake protocol Finished value

Parameters

- **version** – TLS protocol version tuple
- **masterSecret** – negotiated master secret of the connection
- **cipherSuite** – negotiated cipher suite of the connection,
- **handshakeHashes** – running hash of the handshake messages

- **isClient** – whether the calculation should be performed for message sent by client (True) or by server (False) side of connection

tlslite.mathtls.**calcMasterSecret**(*version*, *cipherSuite*, *premasterSecret*, *clientRandom*, *serverRandom*)
Derive Master Secret from premaster secret and random values

tlslite.mathtls.**calc_key**(*version*, *secret*, *cipher_suite*, *label*, *handshake_hashes=None*, *client_random=None*, *server_random=None*, *output_length=None*)
Method for calculating different keys depending on input. It can be used to calculate finished value, master secret, extended master secret or key expansion.

Parameters

- **version** (*tuple(int, int)*) – TLS protocol version
- **secret** (*bytearray*) – master secret or premasterSecret which will be used in the PRF.
- **cipher_suite** (*int*) – Negotiated cipher suite of the connection.
- **label** (*bytes*) – label for the key you want to calculate (ex. 'master secret', 'extended master secret', etc).
- **handshake_hashes** (HandshakeHashes) – running hash of the handshake messages needed for calculating extended master secret or finished value.
- **client_random** (*bytearray*) – client random needed for calculating master secret or key expansion.
- **server_random** (*bytearray*) – server random needed for calculating master secret or key expansion.
- **output_length** (*int*) – Number of bytes to output.

tlslite.mathtls.**createHMAC**(*k*, *digestmod=<built-in function openssl_sha1>*)

tlslite.mathtls.**createMAC_SSL**(*k*, *digestmod=None*)

tlslite.mathtls.**makeK**(*N*, *g*)

tlslite.mathtls.**makeU**(*N*, *A*, *B*)

tlslite.mathtls.**makeVerifier**(*username*, *password*, *bits*)

tlslite.mathtls.**makeX**(*salt*, *username*, *password*)

tlslite.mathtls.**paramStrength**(*param*)
Return level of security for DH, DSA and RSA parameters.

Provide the approximate level of security for algorithms based on finite field (DSA, DH) or integer factorisation cryptography (RSA) when provided with the prime defining the field or the modulus of the public key.

Parameters **param** (*int*) – prime or modulus

tlslite.messages module

Classes representing TLS messages.

class tlslite.messages.**Alert**
Bases: object

__init__()
Initialize self. See help(type(self)) for accurate signature.

__repr__()
Return repr(self).

__str__()
Return str(self).

create(*description*, *level=2*)

descriptionName

levelName

parse(*p*)

write()

class tlslite.messages.**ApplicationData**
Bases: object

__init__()
Initialize self. See help(type(self)) for accurate signature.

create(*bytes*)

parse(*p*)

splitFirstByte()

write()

class tlslite.messages.**Certificate**(*certificateType*, *version=(3, 2)*)
Bases: *tlslite.messages.HandshakeMsg*

__init__(*certificateType*, *version=(3, 2)*)
Initialize self. See help(type(self)) for accurate signature.

__repr__()
Return repr(self).

cert_chain
Getter for the cert_chain property.

create(*cert_chain*, *context=b''*)
Initialise fields of the class.

parse(*p*)

write()

class tlslite.messages.**CertificateEntry**(*certificateType*)
Bases: object

Object storing a single certificate from TLS 1.3.

Stores a certificate (or possibly a raw public key) together with associated extensions

__init__(*certificateType*)
Initialise the object for given certificate type.

__repr__()
Return repr(self).

create(*certificate*, *extensions*)
Set all values of the certificate entry.

parse(*parser*)
Deserialise the object from on the wire data.

write()
Serialise the object.

class tlslite.messages.**CertificateRequest**(*version*)
Bases: *tlslite.messages.HelloMessage*

__init__(*version*)
Initialize object.

create(*certificate_types=None*, *certificate_authorities=None*, *sig_algs=None*, *context=b''*, *extensions=None*)
Creates a Certificate Request message. For TLS 1.3 only the context and extensions parameters should be provided, the others are ignored. For TLS versions below 1.3 instead only the first three parameters are considered.

parse(*parser*)

supported_signature_algs
Returns the list of supported algorithms.

We store the list in an extension even for TLS < 1.3 Extensions are used/valid only for TLS 1.3 but they are a good unified storage mechanism for all versions.

write()

class tlslite.messages.**CertificateStatus**
Bases: *tlslite.messages.HandshakeMsg*

Handling of the CertificateStatus message from RFC 6066.

Handling of the handshake protocol message that includes the OCSP staple.

Variables

- **status_type** (*int*) – type of response returned
- ***ocsp*** (*bytearray*) – OCSPResponse from RFC 2560

__init__()
Create the objet, set its type.

create(*status_type*, *ocsp*)
Set up message payload.

parse(*parser*)
Deserialise the message from one the wire data.

write()
Serialise the message.

class tlslite.messages.**CertificateVerify**(*version*)
Bases: *tlslite.messages.HandshakeMsg*

Serializer for TLS handshake protocol Certificate Verify message.

__init__(*version*)
Create message.

Parameters **version** – TLS protocol version in use

create(*signature*, *signatureAlgorithm=None*)
Provide data for serialisation of message.

Parameters

- **signature** – signature carried in the message

- **signatureAlgorithm** – signature algorithm used to make the signature (TLSv1.2 only)

parse(*parser*)
Deserialize message from parser.

Parameters **parser** – parser with data to read

write()
Serialize the data to bytearray.

Return type bytearray

class tlslite.messages.**ChangeCipherSpec**
Bases: object

__init__()
Initialize self. See help(type(self)) for accurate signature.

create()

parse(*p*)

write()

class tlslite.messages.**ClientFinished**
Bases: *tlslite.messages.SSL2Finished*

Handling of SSLv2 CLIENT-FINISHED message.

Variables **verify_data** (*bytearray*) – payload of the message, should be the CONNECTION-ID

__init__()
Initialize self. See help(type(self)) for accurate signature.

class tlslite.messages.**ClientHello**(*ssl2=False*)
Bases: *tlslite.messages.HelloMessage*

Class for handling the ClientHello SSLv2/SSLv3/TLS message.

Variables

- ***certificate_types*** (*list*) – list of supported certificate types (deprecated)
- ***srp_username*** (*bytearray*) – name of the user in SRP extension (deprecated)
- **supports_npn** (*boolean*) – NPN extension presence (deprecated)
- **tack** (*boolean*) – TACK extension presence (deprecated)
- **server_name** (*bytearray*) – first host_name (type 0) present in SNI extension (deprecated)
- ***extensions*** (list of TLSExtension) – list of TLS extensions parsed from wire or to send, see TLSExtension and child classes for exact examples

__init__(*ssl2=False*)
Initialize object.

__repr__()
Return machine readable representation of Client Hello.

Return type str

__str__()
Return human readable representation of Client Hello.

Return type str

certificate_types
Return the list of certificate types supported.

Deprecated since version 0.5: use extensions field to get the extension for inspection

create(*version*, *random*, *session_id*, *cipher_suites*, *certificate_types=None*, *srpUsername=None*, *tack=False*, *supports_npn=None*, *serverName=None*, *extensions=None*)
Create a ClientHello message for sending.

Parameters

- **version** (*tuple*) – the highest supported TLS version encoded as two int tuple
- **random** (*bytearray*) – client provided random value, in old versions of TLS (before 1.2) the first 32 bits should include system time, also used as the "challenge" field in SSLv2
- **session_id** (*bytearray*) – ID of session, set when doing session resumption
- **cipher_suites** (*list*) – list of ciphersuites advertised as supported
- **certificate_types** (*list*) – list of supported certificate types, uses TLS extension for signalling, as such requires TLS1.0 to work
- **srpUsername** (*bytearray*) – utf-8 encoded username for SRP, TLS extension
- **tack** (*boolean*) – whatever to advertise support for TACK, TLS extension
- **supports_npn** (*boolean*) – whatever to advertise support for NPN, TLS extension
- **serverName** (*bytearray*) – the hostname to request in server name indication extension, TLS extension. Note that SNI allows to set multiple hostnames and values that are not hostnames, use *SNIExtension* together with `extensions` to use it.
- **extensions** (list of *TLSExtension*) – list of extensions to advertise

parse(*p*)
Deserialise object from on the wire data.

psk_truncate()
Return a truncated encoding of message without binders.

In TLS 1.3, with PSK exchange, the ClientHello message is signed by the binders in it. Return the part that is symmetrically signed by those binders.

See "PSK Binder" in draft-ietf-tls-tls13-23.

Return type bytearray

server_name
Return first host_name present in SNI extension.

Deprecated since version 0.5: use extensions field to get the extension for inspection

Return type bytearray

srp_username
Return username for the SRP.

Deprecated since version 0.5: use extensions field to get the extension for inspection

supports_npn
Return whether client supports NPN extension.

Deprecated since version 0.5: use extensions field to get the extension for inspection

Return type boolean

tack
Return whether the client supports TACK.

Deprecated since version 0.5: use extensions field to get the extension for inspection

Return type boolean

write()
Serialise object to on the wire data.

class tlslite.messages.**ClientKeyExchange**(*cipherSuite*, *version=None*)
Bases: *tlslite.messages.HandshakeMsg*

Handling of TLS Handshake protocol ClientKeyExchange message.

Variables

- **cipherSuite** (*int*) – the cipher suite id used for the connection
- **version** (*tuple(int, int)*) – TLS protocol version used for the connection
- **srp_A** (*int*) – SRP protocol client answer value
- **dh_Yc** (*int*) – client Finite Field Diffie-Hellman protocol key share
- **ecdh_Yc** (*bytearray*) – encoded curve coordinates
- **encryptedPreMasterSecret** (*bytearray*) – client selected PremMaster secret encrypted with server public key (from certificate)

__init__(*cipherSuite*, *version=None*)
Initialise ClientKeyExchange for reading or writing.

Parameters

- **cipherSuite** (*int*) – id of the ciphersuite selected by server
- **version** (*tuple(int, int)*) – protocol version selected by server

createDH(*dh_Yc*)
Set the client FFDH key share.

returns self

Return type *ClientKeyExchange*

createECDH(*ecdh_Yc*)
Set the client ECDH key share.

returns self

Return type *ClientKeyExchange*

createRSA(*encryptedPreMasterSecret*)
Set the encrypted PreMaster Secret.

returns self

Return type *ClientKeyExchange*

createSRP(*srp_A*)
Set the SRP client answer.

returns self

Parameters **srp_A** (*int*) – client SRP answer

Return type *ClientKeyExchange*

parse(*parser*)
Deserialise the message from `Parser`,

returns self

Return type *ClientKeyExchange*

write()
Serialise the object.

Return type bytearray

class tlslite.messages.**ClientMasterKey**
Bases: *tlslite.messages.HandshakeMsg*

Handling of SSLv2 CLIENT-MASTER-KEY message.

Variables

- **cipher** (*int*) – negotiated cipher
- **clear_key** (*bytearray*) – the part of master secret key that is sent in clear for export cipher suites
- **encrypted_key** (*bytearray*) – (part of) master secret encrypted using server key
- **key_argument** (*bytearray*) – additional key argument for block ciphers

__init__()
Initialize self. See help(type(self)) for accurate signature.

create(*cipher*, *clear_key*, *encrypted_key*, *key_argument*)
Set values of the CLIENT-MASTER-KEY object.

parse(*parser*)
Deserialise object from on the wire data.

write()
Serialise the object to on the wire data.

class tlslite.messages.**EncryptedExtensions**
Bases: *tlslite.messages.HelloMessage*

Handling of the TLS1.3 Encrypted Extensions message.

__init__()
Initialize object.

create(*extensions*)
Set the extensions in the message.

parse(*parser*)
Parse the extensions from on the wire data.

write()
Serialise the message to on the wire data.

Return type bytearray

class tlslite.messages.**Finished**(*version*, *hash_length=None*)
Bases: *tlslite.messages.HandshakeMsg*

__init__(*version*, *hash_length=None*)
Initialize self. See help(type(self)) for accurate signature.

create(*verify_data*)

parse(*p*)

write()

class tlslite.messages.**HandshakeMsg**(*handshakeType*)
Bases: object

__init__(*handshakeType*)
Initialize self. See help(type(self)) for accurate signature.

postWrite(*w*)

class tlslite.messages.**Heartbeat**
Bases: object

Handling Heartbeat messages from RFC 6520

Variables

- **message_type** – type of message (response or request)
- **payload** – payload
- **padding** – random padding of selected length

__init__()
Initialize self. See help(type(self)) for accurate signature.

__str__()
Return human readable representation of heartbeat message.

create(*message_type*, *payload*, *padding_length*)
Create heartbeat request or response with selected parameters

create_response()
Creates heartbeat response based on request.

parse(*p*)
Deserialize heartbeat message from parser.

We are reading only message type and payload, ignoring leftover bytes (padding).

write()
Serialise heartbeat message.

class tlslite.messages.**HelloMessage**(**args*, ***kwargs*)
Bases: *tlslite.messages.HandshakeMsg*

Class for sharing code between *ClientHello* and *ServerHello*.

__init__(**args*, ***kwargs*)
Initialize object.

addExtension(*ext*)
Add extension to internal list of extensions.

Parameters **ext** (TLSExtension) – extension object to add to list

getExtension(*extType*)
Return extension of given type if present, None otherwise.

Return type *TLSExtension*

Raises ***TLSInternalError*** – when there are multiple extensions of the same type

class tlslite.messages.**HelloRequest**
Bases: *tlslite.messages.HandshakeMsg*

Handling of Hello Request messages.

__init__()
Initialize self. See help(type(self)) for accurate signature.

create()

parse(*parser*)

write()

class tlslite.messages.**KeyUpdate**
Bases: *tlslite.messages.HandshakeMsg*

Handling KeyUpdate message from RFC 8446

Variables **message_type** (*int*) – type of message (update_not_requested or update_requested)

__init__()
Initialize self. See help(type(self)) for accurate signature.

create(*message_type*)
Create KeyUpdate message with selected parameter.

parse(*p*)
Deserialize keyupdate message from parser.

write()
Serialise keyupdate message.

class tlslite.messages.**Message**(*contentType*, *data*)
Bases: object

Generic TLS message.

__init__(*contentType*, *data*)
Initialize object with specified contentType and data.

Parameters

- **contentType** (*int*) – TLS record layer content type of associated data
- **data** (*bytearray*) – data

write()
Return serialised object data.

class tlslite.messages.**NewSessionTicket**
Bases: *tlslite.messages.HelloMessage*

Handling of the TLS1.3 New Session Ticket message.

__init__()
Create New Session Ticket object.

create(*ticket_lifetime*, *ticket_age_add*, *ticket_nonce*, *ticket*, *extensions*)
Initialise a New Session Ticket.

parse(*parser*)
Parse the object from on the wire data.

write()
Serialise the message to on the wire data.

Return type bytearray

class tlslite.messages.**NextProtocol**
Bases: *tlslite.messages.HandshakeMsg*

__init__()
Initialize self. See help(type(self)) for accurate signature.

create(*next_proto*)

parse(*p*)

write(*trial=False*)

class tlslite.messages.**RecordHeader**(*ssl2*)
Bases: object

Generic interface to SSLv2 and SSLv3 (and later) record headers.

__init__(*ssl2*)
Define instance variables.

class tlslite.messages.**RecordHeader2**
Bases: *tlslite.messages.RecordHeader*

SSLv2 record header.

Variables

- **padding** (*int*) – number of bytes added at end of message to make it multiple of block cipher size
- **securityEscape** (*boolean*) – whether the record contains a security escape message

__init__()
Define a SSLv2 style class.

create(*length*, *padding=0*, *securityEscape=False*)
Set object's values.

parse(*parser*)
Deserialise object from Parser.

write()
Serialise object to bytearray.

class tlslite.messages.**RecordHeader3**
Bases: *tlslite.messages.RecordHeader*

SSLv3 (and later) TLS record header.

__init__()
Define a SSLv3 style class.

__repr__()
Return repr(self).

__str__()
Return str(self).

create(*version*, *type*, *length*)
Set object values for writing (serialisation).

parse(*parser*)
Deserialise object from Parser.

typeName

write()
Serialise object to bytearray.

class tlslite.messages.**SSL2Finished**(*msg_type*)
Bases: *tlslite.messages.HandshakeMsg*

Handling of the SSL2 FINISHED messages.

__init__(*msg_type*)
Initialize self. See help(type(self)) for accurate signature.

create(*verify_data*)
Set the message payload.

parse(*parser*)
Deserialise the message from on the wire data.

write()
Serialise the message to on the wire data.

class tlslite.messages.**ServerFinished**
Bases: *tlslite.messages.SSL2Finished*

Handling of SSLv2 SERVER-FINISHED message.

Variables **verify_data** (*bytearray*) – payload of the message, should be SESSION-ID

__init__()
Initialize self. See help(type(self)) for accurate signature.

class tlslite.messages.**ServerHello**
Bases: *tlslite.messages.HelloMessage*

Handling of Server Hello messages.

Variables

- **server_version** (*tuple*) – protocol version encoded as two int tuple
- **random** (*bytearray*) – server random value
- **session_id** (*bytearray*) – session identifier for resumption
- **cipher_suite** (*int*) – server selected cipher_suite
- **compression_method** (*int*) – server selected compression method
- ***next_protos*** (*list of bytearray*) – list of advertised protocols in NPN extension
- ***next_protos_advertised*** (*list of bytearray*) – list of protocols advertised in NPN extension
- ***certificate_type*** (*int*) – certificate type selected by server
- ***extensions*** (*list*) – list of TLS extensions present in server_hello message, see *TLSExtension* and child classes for exact examples

__init__()
Initialise ServerHello object.

__repr__()
Return repr(self).

__str__()
Return str(self).

certificate_type
Return the certificate type selected by server.

Return type int

create(*version*, *random*, *session_id*, *cipher_suite*, *certificate_type=None*, *tackExt=None*, *next_protos_advertised=None*, *extensions=None*)
Initialize the object for deserialisation.

next_protos
Return the advertised protocols in NPN extension.

Return type list of bytearrays

next_protos_advertised
Return the advertised protocols in NPN extension.

Return type list of bytearrays

parse(*p*)

tackExt
Return the TACK extension.

write()

class tlslite.messages.**ServerHello2**
Bases: *tlslite.messages.HandshakeMsg*

SERVER-HELLO message from SSLv2.

Variables

- **session_id_hit** (*int*) – non zero if the client provided session ID was matched in server's session cache
- ***certificate_type*** (*int*) – type of certificate sent
- **server_version** (*tuple of ints*) – protocol version selected by server
- ***certificate*** (*bytearray*) – certificate sent by server
- **ciphers** (*array of int*) – list of ciphers supported by server
- **session_id** (*bytearray*) – idendifier of negotiated session

__init__()
Initialize self. See help(type(self)) for accurate signature.

create(*session_id_hit*, *certificate_type*, *server_version*, *certificate*, *ciphers*, *session_id*)
Initialize fields of the SERVER-HELLO message.

parse(*parser*)
Deserialise object from on the wire data.

write()
Serialise object to on the wire data.

class tlslite.messages.**ServerHelloDone**
Bases: *tlslite.messages.HandshakeMsg*

__init__()
Initialize self. See help(type(self)) for accurate signature.

__repr__()
Human readable representation of object.

create()

parse(*p*)

write()

class tlslite.messages.**ServerKeyExchange**(*cipherSuite*, *version*)
Bases: *tlslite.messages.HandshakeMsg*

Handling TLS Handshake protocol Server Key Exchange messages.

Variables

- **cipherSuite** (*int*) – id of ciphersuite selected in Server Hello message
- **srp_N** (*int*) – SRP protocol prime
- **srp_N_len** (*int*) – length of srp_N in bytes
- **srp_g** (*int*) – SRP protocol generator
- **srp_g_len** (*int*) – length of srp_g in bytes
- **srp_s** (*bytearray*) – SRP protocol salt value
- **srp_B** (*int*) – SRP protocol server public value
- **srp_B_len** (*int*) – length of srp_B in bytes
- **dh_p** (*int*) – FFDHE protocol prime
- **dh_p_len** (*int*) – length of dh_p in bytes
- **dh_g** (*int*) – FFDHE protocol generator
- **dh_g_len** (*int*) – length of dh_g in bytes
- **dh_Ys** (*int*) – FFDH protocol server key share
- **dh_Ys_len** (*int*) – length of dh_Ys in bytes
- **curve_type** (*int*) – Type of curve used (explicit, named, etc.)
- ***named_curve*** (*int*) – TLS ID of named curve
- **ecdh_Ys** (*bytearray*) – ECDH protocol encoded point key share
- **signature** (*bytearray*) – signature performed over the parameters by server
- **hashAlg** (*int*) – id of hash algorithm used for signature
- **signAlg** (*int*) – id of signature algorithm used for signature

__init__(*cipherSuite*, *version*)
Initialise Server Key Exchange for reading or writing.

Parameters **cipherSuite** (*int*) – id of ciphersuite selected by server

__repr__()
Return repr(self).

createDH(*dh_p*, *dh_g*, *dh_Ys*)
Set FFDH protocol parameters.

createECDH(*curve_type*, *named_curve=None*, *point=None*)
Set ECDH protocol parameters.

createSRP(*srp_N*, *srp_g*, *srp_s*, *srp_B*)
Set SRP protocol parameters.

hash(*clientRandom*, *serverRandom*)
Calculate hash of parameters to sign.

Return type bytearray

parse(*parser*)
Deserialise message from `Parser`.

Parameters **parser** (`Parser`) – parser to read data from

write()
Serialise complete message.

Return type bytearray

writeParams()
Serialise the key exchange parameters.

Return type bytearray

class tlslite.messages.**SessionTicketPayload**
Bases: `object`

Serialisation and deserialisation of server state for resumption.

This is the internal (meant to be encrypted) representation of server state that is set to client in the NewSessionTicket message.

Variables

- **version** (*int*) – implementation detail for forward compatibility
- **master_secret** (*bytearray*) – master secret for TLS 1.2-, resumption master secret for TLS 1.3
- ***protocol_version*** (*tuple*) – version of protocol that was previously negotiated in this session
- **cipher_suite** (*int*) – numerical ID of ciphersuite that was negotiated previously
- **nonce** (*bytearray*) – nonce for TLS 1.3 KDF
- **creation_time** (*int*) – Unix time in seconds when was the ticket created
- ***client_cert_chain*** (`X509CertChain`) – Client X509 Certificate Chain

__init__()
Create instance of the object.

client_cert_chain
Getter for the client_cert_chain property.

create(*master_secret*, *protocol_version*, *cipher_suite*, *creation_time*, *nonce=bytearray(b'')*, *client_cert_chain=None*)
Initialise the object with cryptographic data.

parse(*parser*)

write()

tlslite.messages.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.messagesocket module

Wrapper of TLS RecordLayer providing message-level abstraction

class tlslite.messagesocket.**MessageSocket**(*sock*, *defragmenter*)
Bases: *tlslite.recordlayer.RecordLayer*

TLS Record Layer socket that provides Message level abstraction

Because the record layer has a hard size limit on sent messages, they need to be fragmented before sending. Similarly, a single record layer record can include multiple handshake protocol messages (very common with ServerHello, Certificate and ServerHelloDone), as such, the user of RecordLayer needs to fragment those records into multiple messages. Unfortunately, fragmentation of messages requires some degree of knowledge about the messages passed and as such is outside scope of pure record layer implementation.

This class tries to provide a useful abstraction for handling Handshake protocol messages.

Variables

- ***recordSize*** (*int*) – maximum size of records sent through socket. Messages bigger than this size will be fragmented to smaller chunks. Setting it to higher value than the default 2^14 will make the implementation non RFC compliant and likely not interoperable with other peers.
- ***defragmenter*** (Defragmenter) – defragmenter used for read records
- **unfragmentedDataTypes** (*tuple*) – data types which will be passed as-read, TLS application_data and heartbeat by default

__init__(*sock*, *defragmenter*)
Apply TLS Record Layer abstraction to raw network socket.

Parameters

- **sock** (*socket.socket*) – network socket to wrap
- **defragmenter** (Defragmenter) – defragmenter to apply on the records read

flush()
Empty the queue of messages to write

Will fragment the messages and write them in as little records as possible.

Return type generator

flushBlocking()
Blocking variant of *flush()*.

queueMessage(*msg*)
Queue message for sending

If the message is of same type as messages in queue, the message is just added to queue.

If the message is of different type as messages in queue, the queue is flushed and then the message is queued.

Return type generator

queueMessageBlocking(*msg*)
Blocking variant of *queueMessage()*.

recvMessage()
Read next message in queue

will return a 0 or 1 if the read is blocking, a tuple of RecordHeader3 and Parser in case a message was received.

Return type generator

recvMessageBlocking()
Blocking variant of *recvMessage()*.

sendMessage(*msg*)
Fragment and send a message.

If a messages already of same type reside in queue, the message if first added to it and then the queue is flushed.

If the message is of different type than the queue, the queue is flushed, the message is added to queue and the queue is flushed again.

Use the sendRecord() message if you want to send a message outside the queue, or a message of zero size.

Return type generator

sendMessageBlocking(*msg*)
Blocking variant of *sendMessage()*.

tlslite.recordlayer module

Implementation of the TLS Record Layer protocol

class tlslite.recordlayer.**ConnectionState**
Bases: object

Preserve the connection state for reading and writing data to records

__init__()
Create an instance with empty encryption and MACing contexts

getSeqNumBytes()
Return encoded sequence number and increment it.

class tlslite.recordlayer.**RecordLayer**(*sock*)
Bases: object

Implementation of TLS record layer protocol

Variables

- **version** – the TLS version to use (tuple encoded as on the wire)
- **sock** – underlying socket
- **client** – whether the connection should use encryption
- **handshake_finished** – used in SSL2, True if handshake protocol is over
- ***tls13record*** – if True, the record layer will use the TLS 1.3 version and content type hiding
- ***early_data_ok*** (*bool*) – if True, it's ok to ignore undecryptable records up to the size of max_early_data (sum of payloads)

- **max_early_data** (*int*) – maximum number of bytes that will be processed before aborting the connection on data that can not be validated, works only if early_data_ok is set to True
- **padding_cb** (*callable*) – callback used for calculating the size of padding to add in TLSv1.3 records
- **send_record_limit** (*int*) – hint provided to padding callback to not generate records larger than the receiving size expects
- ***recv_record_limit*** (*int*) – negotiated size of records we are willing to accept, TLSRecordOverflow will be raised when records with larger plaintext size are received (in TLS 1.3 padding is included in this size but encrypted content type is not)

__init__(*sock*)
Initialize self. See help(type(self)) for accurate signature.

addPadding(*data*)
Add padding to data so that it is multiple of block size

blockSize
Return the size of block used by current symmetric cipher (R/O)

calcPendingStates(*cipherSuite*, *masterSecret*, *clientRandom*, *serverRandom*, *implementations*)
Create pending states for encryption and decryption.

calcSSL2PendingStates(*cipherSuite*, *masterSecret*, *clientRandom*, *serverRandom*, *implementations*)
Create the keys for encryption and decryption in SSLv2

While we could reuse calcPendingStates(), we need to provide the key-arg data for the server that needs to be passed up to handshake protocol.

calcTLS1_3KeyUpdate_reciever(*cipherSuite*, *cl_app_secret*, *sr_app_secret*)

calcTLS1_3KeyUpdate_sender(*cipherSuite*, *cl_app_secret*, *sr_app_secret*)

calcTLS1_3PendingState(*cipherSuite*, *cl_traffic_secret*, *sr_traffic_secret*, *implementations*)
Create pending state for encryption in TLS 1.3.

Parameters

- **cipherSuite** (*int*) – cipher suite that will be used for encrypting and decrypting data
- **cl_traffic_secret** (*bytearray*) – Client Traffic Secret, either handshake secret or application data secret
- **sr_traffic_secret** (*bytearray*) – Server Traffic Secret, either handshake secret or application data secret
- **implementations** (*list*) – list of names of implementations that are permitted for the connection

calculateMAC(*mac*, *seqnumBytes*, *contentType*, *data*)
Calculate the SSL/TLS version of a MAC

changeReadState()
Change the cipher state to the pending one for read operations.

This should be done only once after a call to *calcPendingStates()* was performed and directly after receiving a ChangeCipherSpec message.

changeWriteState()
Change the cipher state to the pending one for write operations.

This should be done only once after a call to *calcPendingStates()* was performed and directly after sending a ChangeCipherSpec message.

early_data_ok
Set or get the state of early data acceptability.

If processing of the early_data records is to suceed, even if the encryption is not correct, set this property to True. It will be automatically reset to False as soon as a decryptable record is processed.

Use max_early_data to set the limit of the total size of records that will be processed like this.

encryptThenMAC
Set or get the setting of Encrypt Then MAC mechanism.

set the encrypt-then-MAC mechanism for record integrity for next parameter change (after CCS), gets current state

getCipherImplementation()
Return the name of the implementation used for the connection

'python' for tlslite internal implementation, 'openssl' for M2crypto and 'pycrypto' for pycrypto :rtype: str :returns: Name of cipher implementation used, None if not initialised

getCipherName()
Return the name of the bulk cipher used by this connection

Return type str

Returns The name of the cipher, like 'aes128', 'rc4', etc.

isCBCMode()
Returns true if cipher uses CBC mode

recvRecord()
Read, decrypt and check integrity of a single record

Return type tuple

Returns message header and decrypted message payload

Raises

- ***TLSDecryptionFailed*** – when decryption of data failed
- ***TLSBadRecordMAC*** – when record has bad MAC or padding
- **socket.error** – when reading from socket was unsuccessful
- ***TLSRecordOverflow*** – when the received record was longer than allowed by negotiated version of TLS

recv_record_limit
Maximum record size that is permitted for receiving.

sendRecord(*msg*)
Encrypt, MAC and send arbitrary message as-is through socket.

Note that if the message was not fragmented to below 2**14 bytes it will be rejected by the other connection side.

Parameters **msg** (ApplicationData, *HandshakeMessage, etc.*) – TLS message to send

shutdown()
Clear read and write states

tls13record
Return the value of the tls13record state.

version
Return the TLS version used by record layer

class tlslite.recordlayer.**RecordSocket**(*sock*)
Bases: object

Socket wrapper for reading and writing TLS Records.

Variables

- **sock** – wrapped socket
- **version** – version for the records to be encoded on the wire
- ***tls13record*** – flag to indicate that TLS 1.3 specific record limits should be used for received records
- ***recv_record_limit*** (*int*) – negotiated maximum size of record plaintext size

__init__(*sock*)
Assign socket to wrapper

recv()
Read a single record from socket, handle SSLv2 and SSLv3 record layer

Return type generator

Returns generator that returns 0 or 1 in case the read would be blocking or a tuple containing record header (object) and record data (bytearray) read from socket

Raises

- **socket.error** – In case of network error
- ***TLSAbruptCloseError*** – When the socket was closed on the other side in middle of record receiving
- ***TLSRecordOverflow*** – When the received record was longer than allowed by TLS
- ***TLSIllegalParameterException*** – When the record header was malformed

send(*msg*, *padding=0*)
Send the message through socket.

Parameters

- **msg** (*bytearray*) – TLS message to send
- **padding** (*int*) – amount of padding to specify for SSLv2

Raises **socket.error** – when write to socket failed

tlslite.session module

Class representing a TLS session.

class tlslite.session.**Session**
Bases: object

This class represents a TLS session.

TLS distinguishes between connections and sessions. A new handshake creates both a connection and a session. Data is transmitted over the connection.

The session contains a more permanent record of the handshake. The session can be inspected to determine handshake results. The session can also be used to create a new connection through "session resumption". If the client and server both support this, they can create a new connection based on an old session without the overhead of a full handshake.

The session for a *TLSConnection* can be retrieved from the connection's 'session' attribute.

Variables

- **srpUsername** (*str*) – The client's SRP username (or None).
- **clientCertChain** (X509CertChain) – The client's certificate chain (or None).
- **serverCertChain** (X509CertChain) – The server's certificate chain (or None).
- ***tackExt*** (*tack.structures.TackExtension.TackExtension*) – The server's TackExtension (or None).
- **tackInHelloExt** (*bool*) – True if a TACK was presented via TLS Extension.
- **encryptThenMAC** (*bool*) – True if connection uses CBC cipher in encrypt-then-MAC mode
- **appProto** (*bytearray*) – name of the negotiated application level protocol, None if not negotiated
- **cl_app_secret** (*bytearray*) – key used for deriving keys used by client to encrypt and protect data in TLS 1.3
- **sr_app_secret** (*bytearray*) – key used for deriving keys used by server to encrypt and protect data in TLS 1.3
- **exporterMasterSecret** (*bytearray*) – master secret used for TLS Exporter in TLS1.3
- **resumptionMasterSecret** (*bytearray*) – master secret used for session resumption in TLS 1.3
- **tickets** (*list*) – list of tickets received from the server

__init__()
Initialize self. See help(type(self)) for accurate signature.

create(*masterSecret*, *sessionID*, *cipherSuite*, *srpUsername*, *clientCertChain*, *serverCertChain*, *tackExt*, *tackInHelloExt*, *serverName*, *resumable=True*, *encryptThenMAC=False*, *extendedMasterSecret=False*, *appProto=bytearray(b'')*, *cl_app_secret=bytearray(b'')*, *sr_app_secret=bytearray(b'')*, *exporterMasterSecret=bytearray(b'')*, *resumptionMasterSecret=bytearray(b'')*, *tickets=None*)

getBreakSigs()

getCipherName()
Get the name of the cipher used with this connection.

Return type str

Returns The name of the cipher used with this connection.

getMacName()
Get the name of the HMAC hash algo used with this connection.

Return type str

Returns The name of the HMAC hash algo used with this connection.

getTackId()

valid()
If this session can be used for session resumption.

Return type bool

Returns If this session can be used for session resumption.

tlslite.session.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.sessioncache module

Class for caching TLS sessions.

class tlslite.sessioncache.**SessionCache**(*maxEntries=10000*, *maxAge=14400*)
Bases: object

This class is used by the server to cache TLS sessions.

Caching sessions allows the client to use TLS session resumption and avoid the expense of a full handshake. To use this class, simply pass a SessionCache instance into the server handshake function.

This class is thread-safe.

__getitem__(*sessionID*)

__init__(*maxEntries=10000*, *maxAge=14400*)
Create a new SessionCache.

Parameters

- **maxEntries** (*int*) – The maximum size of the cache. When this limit is reached, the oldest sessions will be deleted as necessary to make room for new ones. The default is 10000.
- **maxAge** (*int*) – The number of seconds before a session expires from the cache. The default is 14400 (i.e. 4 hours).

__setitem__(*sessionID*, *session*)

tlslite.tlsconnection module

MAIN CLASS FOR TLS LITE (START HERE!).

class tlslite.tlsconnection.**TLSConnection**(*sock*)
Bases: *tlslite.tlsrecordlayer.TLSRecordLayer*

This class wraps a socket and provides TLS handshaking and data transfer.

To use this class, create a new instance, passing a connected socket into the constructor. Then call some handshake function. If the handshake completes without raising an exception, then a TLS connection has been negotiated. You can transfer data over this connection as if it were a socket.

This class provides both synchronous and asynchronous versions of its key functions. The synchronous versions should be used when writing single-or multi-threaded code using blocking sockets. The asynchronous versions should be used when performing asynchronous, event-based I/O with non-blocking sockets.

Asynchronous I/O is a complicated subject; typically, you should not use the asynchronous functions directly, but should use some framework like asyncore or Twisted which TLS Lite integrates with (see *TLSAsyncDispatcherMixIn*).

__init__(*sock*)

Create a new TLSConnection instance.

Parameters **sock** (*socket.socket*) – The socket data will be transmitted on. The socket should already be connected. It may be in blocking or non-blocking mode.

handshakeClientAnonymous(*session=None*, *settings=None*, *checker=None*, *serverName=None*, *async_=False*)

Perform an anonymous handshake in the role of client.

This function performs an SSL or TLS handshake using an anonymous Diffie Hellman ciphersuite.

Like any handshake function, this can be called on a closed TLS connection, or on a TLS connection that is already open. If called on an open connection it performs a re-handshake.

If the function completes without raising an exception, the TLS connection will be open and available for data transfer.

If an exception is raised, the connection will have been automatically closed (if it was ever open).

Parameters

- **session** (Session) – A TLS session to attempt to resume. If the resumption does not succeed, a full handshake will be performed.
- **settings** (HandshakeSettings) – Various settings which can be used to control the ciphersuites, certificate types, and SSL/TLS versions offered by the client.
- **checker** (Checker) – A Checker instance. This instance will be invoked to examine the other party's authentication credentials, if the handshake completes succesfully.
- **serverName** (*string*) – The ServerNameIndication TLS Extension.
- **async** (*bool*) – If False, this function will block until the handshake is completed. If True, this function will return a generator. Successive invocations of the generator will return 0 if it is waiting to read from the socket, 1 if it is waiting to write to the socket, or will raise StopIteration if the handshake operation is completed.

Return type None or an iterable

Returns If 'async_' is True, a generator object will be returned.

Raises

- **socket.error** – If a socket error occurs.
- ***tlslite.errors.TLSAbruptCloseError*** – If the socket is closed without a preceding alert.
- ***tlslite.errors.TLSAlert*** – If a TLS alert is signalled.
- ***tlslite.errors.TLSAuthenticationError*** – If the checker doesn't like the other party's authentication credentials.

handshakeClientCert(*certChain=None*, *privateKey=None*, *session=None*, *settings=None*, *checker=None*, *nextProtos=None*, *reqTack=True*, *serverName=None*, *async_=False*, *alpn=None*)

Perform a certificate-based handshake in the role of client.

This function performs an SSL or TLS handshake. The server will authenticate itself using an X.509 certificate chain. If the handshake succeeds, the server's certificate chain will be stored in the session's serverCertChain attribute. Unless a checker object is passed in, this function does no validation or checking of the server's certificate chain.

If the server requests client authentication, the client will send the passed-in certificate chain, and use the passed-in private key to authenticate itself. If no certificate chain and private key were passed in, the client will attempt to proceed without client authentication. The server may or may not allow this.

If the function completes without raising an exception, the TLS connection will be open and available for data transfer.

If an exception is raised, the connection will have been automatically closed (if it was ever open).

Parameters

- **certChain** (X509CertChain) – The certificate chain to be used if the server requests client authentication.
- **privateKey** (RSAKey) – The private key to be used if the server requests client authentication.
- **session** (Session) – A TLS session to attempt to resume. If the resumption does not succeed, a full handshake will be performed.
- **settings** (HandshakeSettings) – Various settings which can be used to control the ciphersuites, certificate types, and SSL/TLS versions offered by the client.
- **checker** (Checker) – A Checker instance. This instance will be invoked to examine the other party's authentication credentials, if the handshake completes succesfully.
- **nextProtos** (*list of str*) – A list of upper layer protocols ordered by preference, to use in the Next-Protocol Negotiation Extension.
- **reqTack** (*bool*) – Whether or not to send a "tack" TLS Extension, requesting the server return a TackExtension if it has one.
- **serverName** (*string*) – The ServerNameIndication TLS Extension.
- **async** (*bool*) – If False, this function will block until the handshake is completed. If True, this function will return a generator. Successive invocations of the generator will return 0 if it is waiting to read from the socket, 1 if it is waiting to write to the socket, or will raise StopIteration if the handshake operation is completed.
- **alpn** (*list of bytearrays*) – protocol names to advertise to server as supported by client in the Application Layer Protocol Negotiation extension. Example items in the array include b'http/1.1' or b'h2'.

Return type None or an iterable

Returns If '**async_**' is True, a generator object will be returned.

Raises

- **socket.error** – If a socket error occurs.
- ***tlslite.errors.TLSAbruptCloseError*** – If the socket is closed without a preceding alert.
- ***tlslite.errors.TLSAlert*** – If a TLS alert is signalled.

- **tlslite.errors.TLSAuthenticationError** – If the checker doesn't like the other party's authentication credentials.

handshakeClientSRP(*username*, *password*, *session=None*, *settings=None*, *checker=None*, *reqTack=True*, *serverName=None*, *async_=False*)

Perform an SRP handshake in the role of client.

This function performs a TLS/SRP handshake. SRP mutually authenticates both parties to each other using only a username and password. This function may also perform a combined SRP and server-certificate handshake, if the server chooses to authenticate itself with a certificate chain in addition to doing SRP.

If the function completes without raising an exception, the TLS connection will be open and available for data transfer.

If an exception is raised, the connection will have been automatically closed (if it was ever open).

Parameters

- **username** (*bytearray*) – The SRP username.
- **password** (*bytearray*) – The SRP password.
- **session** (Session) – A TLS session to attempt to resume. This session must be an SRP session performed with the same username and password as were passed in. If the resumption does not succeed, a full SRP handshake will be performed.
- **settings** (HandshakeSettings) – Various settings which can be used to control the ciphersuites, certificate types, and SSL/TLS versions offered by the client.
- **checker** (Checker) – A Checker instance. This instance will be invoked to examine the other party's authentication credentials, if the handshake completes succesfully.
- **reqTack** (*bool*) – Whether or not to send a "tack" TLS Extension, requesting the server return a TackExtension if it has one.
- **serverName** (*string*) – The ServerNameIndication TLS Extension.
- **async** (*bool*) – If False, this function will block until the handshake is completed. If True, this function will return a generator. Successive invocations of the generator will return 0 if it is waiting to read from the socket, 1 if it is waiting to write to the socket, or will raise StopIteration if the handshake operation is completed.

Return type None or an iterable

Returns If 'async_' is True, a generator object will be returned.

Raises

- **socket.error** – If a socket error occurs.
- **tlslite.errors.TLSAbruptCloseError** – If the socket is closed without a preceding alert.
- **tlslite.errors.TLSAlert** – If a TLS alert is signalled.
- **tlslite.errors.TLSAuthenticationError** – If the checker doesn't like the other party's authentication credentials.

handshakeServer(*verifierDB=None*, *certChain=None*, *privateKey=None*, *reqCert=False*, *sessionCache=None*, *settings=None*, *checker=None*, *reqCAs=None*, *tacks=None*, *activationFlags=0*, *nextProtos=None*, *anon=False*, *alpn=None*, *sni=None*)

Perform a handshake in the role of server.

This function performs an SSL or TLS handshake. Depending on the arguments and the behavior of the client, this function can perform an SRP, or certificate-based handshake. It can also perform a combined SRP and server-certificate handshake.

Like any handshake function, this can be called on a closed TLS connection, or on a TLS connection that is already open. If called on an open connection it performs a re-handshake. This function does not send a Hello Request message before performing the handshake, so if re-handshaking is required, the server must signal the client to begin the re-handshake through some other means.

If the function completes without raising an exception, the TLS connection will be open and available for data transfer.

If an exception is raised, the connection will have been automatically closed (if it was ever open).

Parameters

- **verifierDB** (VerifierDB) – A database of SRP password verifiers associated with usernames. If the client performs an SRP handshake, the session's srpUsername attribute will be set.
- **certChain** (X509CertChain) – The certificate chain to be used if the client requests server certificate authentication and no virtual host defined in HandshakeSettings matches ClientHello.
- **privateKey** (RSAKey) – The private key to be used if the client requests server certificate authentication and no virtual host defined in HandshakeSettings matches ClientHello.
- **reqCert** (*bool*) – Whether to request client certificate authentication. This only applies if the client chooses server certificate authentication; if the client chooses SRP authentication, this will be ignored. If the client performs a client certificate authentication, the sessions's clientCertChain attribute will be set.
- **sessionCache** (SessionCache) – An in-memory cache of resumable sessions. The client can resume sessions from this cache. Alternatively, if the client performs a full handshake, a new session will be added to the cache.
- **settings** (HandshakeSettings) – Various settings which can be used to control the ciphersuites and SSL/TLS version chosen by the server.
- **checker** (Checker) – A Checker instance. This instance will be invoked to examine the other party's authentication credentials, if the handshake completes succesfully.
- **reqCAs** (*list of bytearray*) – A collection of DER-encoded DistinguishedNames that will be sent along with a certificate request to help client pick a certificates. This does not affect verification.
- **nextProtos** (*list of str*) – A list of upper layer protocols to expose to the clients through the Next-Protocol Negotiation Extension, if they support it. Deprecated, use the *virtual_hosts* in HandshakeSettings.
- **alpn** (*list of bytearray*) – names of application layer protocols supported. Note that it will be used instead of NPN if both were advertised by client. Deprecated, use the *virtual_hosts* in HandshakeSettings.
- **sni** (*bytearray*) – expected virtual name hostname. Deprecated, use the *virtual_hosts* in HandshakeSettings.

Raises

- **socket.error** – If a socket error occurs.
- ***tlslite.errors.TLSAbruptCloseError*** – If the socket is closed without a preceding alert.

- ***tlslite.errors.TLSAlert*** – If a TLS alert is signalled.
- ***tlslite.errors.TLSAuthenticationError*** – If the checker doesn't like the other party's authentication credentials.

handshakeServerAsync(*verifierDB=None*, *certChain=None*, *privateKey=None*, *reqCert=False*, *sessionCache=None*, *settings=None*, *checker=None*, *reqCAs=None*, *tacks=None*, *activationFlags=0*, *nextProtos=None*, *anon=False*, *alpn=None*, *sni=None*)

Start a server handshake operation on the TLS connection.

This function returns a generator which behaves similarly to handshakeServer(). Successive invocations of the generator will return 0 if it is waiting to read from the socket, 1 if it is waiting to write to the socket, or it will raise StopIteration if the handshake operation is complete.

Return type iterable

Returns A generator; see above for details.

keyingMaterialExporter(*label*, *length=20*)

Return keying material as described in RFC 5705

Parameters

- **label** (*bytearray*) – label to be provided for the exporter
- **length** (*int*) – number of bytes of the keying material to export

request_post_handshake_auth(*settings=None*)

Request Post-handshake Authentication from client.

The PHA process is asynchronous, and client may send some data before its certificates are added to Session object. Calling this generator will only request for the new identity of client, it will not wait for it.

tlslite.tlsconnection.**bytes_to_int**()

Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.tlsrecordlayer module

Helper class for TLSConnection.

class tlslite.tlsrecordlayer.**TLSRecordLayer**(*sock*)

Bases: object

This class handles data transmission for a TLS connection.

Its only subclass is *TLSConnection*. We've separated the code in this class from TLSConnection to make things more readable.

Variables

- **sock** (*socket.socket*) – The underlying socket object.

- ***session*** (*Session*) – The session corresponding to this connection. Due to TLS session resumption, multiple connections can correspond to the same underlying session.
- **version** (*tuple*) – The TLS version being used for this connection. (3,0) means SSL 3.0, and (3,1) means TLS 1.0.
- **closed** (*bool*) – If this connection is closed.
- **resumed** (*bool*) – If this connection is based on a resumed session.
- **allegedSrpUsername** (*str or None*) – This is set to the SRP username asserted by the client, whether the handshake succeeded or not. If the handshake fails, this can be inspected to determine if a guessing attack is in progress against a particular user account.
- **closeSocket** (*bool*) – If the socket should be closed when the connection is closed, defaults to True (writable).

  If you set this to True, TLS Lite will assume the responsibility of closing the socket when the TLS Connection is shutdown (either through an error or through the user calling close()). The default is False.
- **ignoreAbruptClose** (*bool*) – If an abrupt close of the socket should raise an error (writable).

  If you set this to True, TLS Lite will not raise a *TLSAbruptCloseError* exception if the underlying socket is unexpectedly closed. Such an unexpected closure could be caused by an attacker. However, it also occurs with some incorrect TLS implementations.

  You should set this to True only if you're not worried about an attacker truncating the connection, and only if necessary to avoid spurious errors. The default is False.
- **encryptThenMAC** (*bool*) – Whether the connection uses the encrypt-then-MAC construct for CBC cipher suites, will be False also if connection uses RC4 or AEAD.
- ***recordSize*** (*int*) – maximum size of data to be sent in a single record layer message. Note that after encryption is established (generally after handshake protocol has finished) the actual amount of data written to network socket will be larger because of the record layer header, padding or encryption overhead. It can be set to low value (so that there is no fragmentation on Ethernet, IP and TCP level) at the beginning of connection to reduce latency and set to protocol max (2**14) to maximise throughput after sending the first few kiB of data. If negotiated, record_size_limit extension may limit it though, causing reading of the variable to return lower value that was initially set. See also: HandshakeSettings.record_size_limit.
- **tickets** (*list of bytearray*) – list of session tickets received from server, oldest first.
- **client_cert_required** (*bool*) – Set to True to make the post-handshake authentication fail when client doesn't provide a certificate in response

__init__(*sock*)
Initialize self. See help(type(self)) for accurate signature.

clearReadBuffer()

clearWriteBuffer()

close()
Close the TLS connection.

This function will block until it has exchanged close_notify alerts with the other party. After doing so, it will shut down the TLS connection. Further attempts to read through this connection will return "". Further attempts to write through this connection will raise ValueError.

If makefile() has been called on this connection, the connection will be not be closed until the connection object and all file objects have been closed.

Even if an exception is raised, the connection will have been closed.

Raises

- **socket.error** – If a socket error occurs.
- ***tlslite.errors.TLSAbruptCloseError*** – If the socket is closed without a preceding alert.
- ***tlslite.errors.TLSAlert*** – If a TLS alert is signalled.

closeAsync()
Start a close operation on the TLS connection.

This function returns a generator which behaves similarly to close(). Successive invocations of the generator will return 0 if it is waiting to read from the socket, 1 if it is waiting to write to the socket, or will raise StopIteration if the close operation has completed.

Return type iterable

Returns A generator; see above for details.

encryptThenMAC
Whether the connection uses Encrypt Then MAC (RFC 7366)

fileno()
Not implement in TLS Lite.

getCipherImplementation()
Get the name of the cipher implementation used with this connection.

Return type str

Returns The name of the cipher implementation used with this connection. Either 'python', 'openssl', or 'pycrypto'.

getCipherName()
Get the name of the cipher used with this connection.

Return type str

Returns The name of the cipher used with this connection. Either 'aes128', 'aes256', 'rc4', or '3des'.

getVersionName()
Get the name of this TLS version.

Return type str

Returns The name of the TLS version used with this connection. Either None, 'SSL 3.0', 'TLS 1.0', 'TLS 1.1', 'TLS 1.2' or 'TLS 1.3'.

getpeername()
Return the remote address to which the socket is connected (socket emulation).

getsockname()
Return the socket's own address (socket emulation).

gettimeout()
Return the timeout associated with socket operations (socket emulation).

makefile(*mode='r'*, *bufsize=-1*)
Create a file object for the TLS connection (socket emulation).

Return type socket._fileobject

read(*max=None*, *min=1*)
Read some data from the TLS connection.

This function will block until at least 'min' bytes are available (or the connection is closed).

If an exception is raised, the connection will have been automatically closed.

Parameters

- **max** (*int*) – The maximum number of bytes to return.
- **min** (*int*) – The minimum number of bytes to return

Return type str

Returns A string of no more than 'max' bytes, and no fewer than 'min' (unless the connection has been closed, in which case fewer than 'min' bytes may be returned).

Raises

- **socket.error** – If a socket error occurs.
- ***tlslite.errors.TLSAbruptCloseError*** – If the socket is closed without a preceding alert.
- ***tlslite.errors.TLSAlert*** – If a TLS alert is signalled.

readAsync(*max=None*, *min=1*)
Start a read operation on the TLS connection.

This function returns a generator which behaves similarly to read(). Successive invocations of the generator will return 0 if it is waiting to read from the socket, 1 if it is waiting to write to the socket, or a string if the read operation has completed.

Return type iterable

Returns A generator; see above for details.

recordSize
Maximum size of the records that will be sent out.

recv(*bufsize*)
Get some data from the TLS connection (socket emulation).

Raises

- **socket.error** – If a socket error occurs.
- ***tlslite.errors.TLSAbruptCloseError*** – If the socket is closed without a preceding alert.
- ***tlslite.errors.TLSAlert*** – If a TLS alert is signalled.

recv_into(*b*)

send(*s*)
Send data to the TLS connection (socket emulation).

Raises **socket.error** – If a socket error occurs.

send_heartbeat_request(*payload*, *padding_length*)
Synchronous version of write_heartbeat function.

Parameters

- **payload** (*bytes*) – Payload, that we want send in request and get at response.

- **padding_length** (*int*) – Length of padding.

Raises **socket.error** – If a socket error occurs.

send_keyupdate_request(*message_type*)
Send a KeyUpdate message.

Parameters **payload** (*int*) – Type of KeyUpdate message.

Raises **socket.error** – If a socket error occurs.

sendall(*s*)
Send data to the TLS connection (socket emulation).

Raises **socket.error** – If a socket error occurs.

setsockopt(*level*, *optname*, *value*)
Set the value of the given socket option (socket emulation).

settimeout(*value*)
Set a timeout on blocking socket operations (socket emulation).

shutdown(*how*)
Shutdown the underlying socket.

unread(*b*)
Add bytes to the front of the socket read buffer for future reading. Be careful using this in the context of select(...): if you unread the last data from a socket, that won't wake up selected waiters, and those waiters may hang forever.

version
Get the SSL protocol version of connection

write(*s*)
Write some data to the TLS connection.

This function will block until all the data has been sent.

If an exception is raised, the connection will have been automatically closed.

Parameters **s** (*str*) – The data to transmit to the other party.

Raises **socket.error** – If a socket error occurs.

writeAsync(*s*)
Start a write operation on the TLS connection.

This function returns a generator which behaves similarly to write(). Successive invocations of the generator will return 1 if it is waiting to write to the socket, or will raise StopIteration if the write operation has completed.

Return type iterable

Returns A generator; see above for details.

write_heartbeat(*payload*, *padding_length*)
Start a write operation of heartbeat_request.

Parameters

- **payload** (*bytes*) – Payload, that we want send in request and get at response.
- **padding_length** (*int*) – Length of padding.

Raises **socket.error** – If a socket error occurs.

tlslite.tlsrecordlayer.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.verifierdb module

Class for storing SRP password verifiers.

class tlslite.verifierdb.**VerifierDB**(*filename=None*)
Bases: *tlslite.basedb.BaseDB*

This class represent an in-memory or on-disk database of SRP password verifiers.

A VerifierDB can be passed to a server handshake to authenticate a client based on one of the verifiers.

This class is thread-safe.

__init__(*filename=None*)
Create a new VerifierDB instance.

Parameters **filename** (*str*) – Filename for an on-disk database, or None for an in-memory database. If the filename already exists, follow this with a call to open(). To create a new on-disk database, follow this with a call to create().

__setitem__(*username*, *verifierEntry*)
Add a verifier entry to the database.

Parameters

- **username** (*str*) – The username to associate the verifier with. Must be less than 256 characters in length. Must not already be in the database.
- **verifierEntry** (*tuple*) – The verifier entry to add. Use *makeVerifier()* to create a verifier entry.

static makeVerifier(*username*, *password*, *bits*)
Create a verifier entry which can be stored in a VerifierDB.

Parameters

- **username** (*str*) – The username for this verifier. Must be less than 256 characters in length.
- **password** (*str*) – The password for this verifier.
- **bits** (*int*) – This values specifies which SRP group parameters to use. It must be one of (1024, 1536, 2048, 3072, 4096, 6144, 8192). Larger values are more secure but slower. 2048 is a good compromise between safety and speed.

Return type tuple

Returns A tuple which may be stored in a VerifierDB.

tlslite.verifierdb.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.x509 module

Class representing an X.509 certificate.

class tlslite.x509.**X509**
Bases: object

This class represents an X.509 certificate.

Variables

- **bytes** (*bytearray*) – The DER-encoded ASN.1 certificate
- **publicKey** (RSAKey) – The subject public key from the certificate.
- **subject** (*bytearray*) – The DER-encoded ASN.1 subject distinguished name.
- **certAlg** (*str*) – algorithm of the public key, "rsa" for RSASSA-PKCS#1 v1.5, "rsa-pss" for RSASSA-PSS, "ecdsa" for ECDSA

__init__()
Create empty certificate object.

getFingerprint()
Get the hex-encoded fingerprint of this certificate.

Return type str

Returns A hex-encoded fingerprint.

parse(*s*)
Parse a PEM-encoded X.509 certificate.

Parameters **s** (*str*) – A PEM-encoded X.509 certificate (i.e. a base64-encoded certificate wrapped with "—–BEGIN CERTIFICATE—–" and "—–END CERTIFICATE—–" tags).

parseBinary(*cert_bytes*)
Parse a DER-encoded X.509 certificate.

Parameters **bytes** (*L{str} (in python2) or L{bytearray} of unsigned bytes*) – A DER-encoded X.509 certificate.

writeBytes()
Serialise object to a DER encoded string.

tlslite.x509.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

tlslite.x509certchain module

Class representing an X.509 certificate chain.

class tlslite.x509certchain.**X509CertChain**(*x509List=None*)
Bases: object

This class represents a chain of X.509 certificates.

Variables **x509List** (*list*) – A list of *tlslite.x509.X509* instances, starting with the end-entity certificate and with every subsequent certificate certifying the previous.

__init__(*x509List=None*)
Create a new X509CertChain.

Parameters **x509List** (*list*) – A list of *tlslite.x509.X509* instances, starting with the end-entity certificate and with every subsequent certificate certifying the previous.

checkTack(*tack*)

getEndEntityPublicKey()
Get the public key from the end-entity certificate.

Return type RSAKey'

getFingerprint()
Get the hex-encoded fingerprint of the end-entity certificate.

Return type str

Returns A hex-encoded fingerprint.

getNumCerts()
Get the number of certificates in this chain.

Return type int

getTackExt()
Get the TACK and/or Break Sigs from a TACK Cert in the chain.

parsePemList(*s*)
Parse a string containing a sequence of PEM certs.

Raise a SyntaxError if input is malformed.

tlslite.x509certchain.**bytes_to_int**()
Return the integer represented by the given array of bytes.

bytes Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use 'sys.byteorder' as the byte order value.

signed Indicates whether two's complement is used to represent the integer.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

Symbols

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X