Twomemo

Release 1.0.3-stable

Tim Henkes (Syndace)

Nov 08, 2022

CONTENTS

1 Installation **3**

2 Getting Started **5**

3 Package: twomemo **7**
- 3.1 Module: etree 7
- 3.2 Module: twomemo 7

Python Module Index **25**

Index **27**

Backend implementation for python-omemo, equipping python-omemo with support for OMEMO under the namespace `urn:xmpp:omemo:2` (casually/jokingly referred to as "twomemo").

CHAPTER

ONE

INSTALLATION

Install the latest release using pip (`pip install twomemo`) or manually from source by running `pip install .` in the cloned repository.

CHAPTER

TWO

GETTING STARTED

No further preparation is required to get started with this backend. Create an instance of *Twomemo* and pass it to python-omemo to equip it with `urn:xmpp:omemo:2` capabilities.

Users of ElementTree can use the helpers in *Module: etree* for their XML serialization/parsing, which is available after installing xmlschema, or by using `pip install twomemo[xml]`. Users of a different XML framework can use the module as a reference to write their own serialization/parsing.

CHAPTER

THREE

PACKAGE: TWOMEMO

3.1 Module: etree

3.2 Module: twomemo

class twomemo.twomemo.**Twomemo**(*storage*, *max_num_per_session_skipped_keys=1000*, *max_num_per_message_skipped_keys=None*)

Bases: Backend

Backend implementation providing OMEMO in the *urn:xmpp:omemo:2* namespace.

Parameters

- **storage** (*Storage*) –
- **max_num_per_session_skipped_keys** (*int*) –
- **max_num_per_message_skipped_keys** (*Optional[int]*) –

__init__(*storage*, *max_num_per_session_skipped_keys=1000*, *max_num_per_message_skipped_keys=None*)

Parameters

- **storage** (Storage) – The storage to store backend-specific data in. Note that all data keys are prefixed with the backend namespace to avoid name clashes between backends.
- **max_num_per_session_skipped_keys** (int) – The maximum number of skipped message keys to keep around per session. Once the maximum is reached, old message keys are deleted to make space for newer ones. Accessible via max_num_per_session_skipped_keys.
- **max_num_per_message_skipped_keys** (Optional[int]) – The maximum number of skipped message keys to accept in a single message. When set to None (the default), this parameter defaults to the per-session maximum (i.e. the value of the max_num_per_session_skipped_keys parameter). This parameter may only be 0 if the per-session maximum is 0, otherwise it must be a number between 1 and the per-session maximum. Accessible via max_num_per_message_skipped_keys.

Return type
None

property namespace: str

Returns: The namespace provided/handled by this backend implementation.

Return type
str

async load_session(*bare_jid*, *device_id*)

Parameters

- **bare_jid** (str) – The bare JID the device belongs to.
- **device_id** (int) – The id of the device.

Return type
Optional[*SessionImpl*]

Returns
The session associated with the device, or *None* if such a session does not exist.

Warning: Multiple sessions for the same device can exist in memory, however only one session per device can exist in storage. Which one of the in-memory sessions is persisted in storage is controlled by calling the *store_session()* method.

async store_session(*session*)

Store a session, overwriting any previously stored session for the bare JID and device id this session belongs to.

Parameters
session (Session) – The session to store.

Return type
None

Returns
Anything, the return value is ignored.

Warning: Multiple sessions for the same device can exist in memory, however only one session per device can exist in storage. Which one of the in-memory sessions is persisted in storage is controlled by calling this method.

async build_session_active(*bare_jid*, *device_id*, *bundle*, *plain_key_material*)

Actively build a session.

Parameters

- **bare_jid** (str) – The bare JID the device belongs to.
- **device_id** (int) – The id of the device.
- **bundle** (Bundle) – The bundle containing the public key material of the other device required for active session building.
- **plain_key_material** (PlainKeyMaterial) – The key material to encrypt for the recipient as part of the initial key exchange/session initiation.

Return type
Tuple[*SessionImpl*, *EncryptedKeyMaterialImpl*]

Returns
The newly built session, the encrypted key material and the key exchange information required by the other device to complete the passive part of session building. The initiation

property of the returned session must return `ACTIVE`. The `key_exchange` property of the returned session must return the information required by the other party to complete its part of the key exchange.

Raises

KeyExchangeFailed – in case of failure related to the key exchange required for session building.

> **Warning:** This method may be called for a device which already has a session. In that case, the original session must remain in storage and must remain loadable via *load_session()*. Only upon calling *store_session()*, the old session must be overwritten with the new one. In summary, multiple sessions for the same device can exist in memory, while only one session per device can exist in storage, which can be controlled using the *store_session()* method.

async build_session_passive(*bare_jid*, *device_id*, *key_exchange*, *encrypted_key_material*)

Passively build a session.

Parameters

- **bare_jid** (`str`) – The bare JID the device belongs to.
- **device_id** (`int`) – The id of the device.
- **key_exchange** (`KeyExchange`) – Key exchange information for the passive session building.
- **encrypted_key_material** (`EncryptedKeyMaterial`) – The key material to decrypt as part of the initial key exchange/session initiation.

Return type

`Tuple[SessionImpl, PlainKeyMaterialImpl]`

Returns

The newly built session and the decrypted key material. Note that the pre key used to initiate this session must somehow be associated with the session, such that *hide_pre_key()* and *delete_pre_key()* can work.

Raises

- **KeyExchangeFailed** – in case of failure related to the key exchange required for session building.
- **DecryptionFailed** – in case of backend-specific failures during decryption of the initial message.

> **Warning:** This method may be called for a device which already has a session. In that case, the original session must remain in storage and must remain loadable via *load_session()*. Only upon calling *store_session()*, the old session must be overwritten with the new one. In summary, multiple sessions for the same device can exist in memory, while only one session per device can exist in storage, which can be controlled using the *store_session()* method.

async encrypt_plaintext(*plaintext*)

Encrypt some plaintext symmetrically.

Parameters

plaintext (`bytes`) – The plaintext to encrypt symmetrically.

Return type
Tuple[*ContentImpl*, *PlainKeyMaterialImpl*]

Returns
The encrypted plaintext aka content, as well as the key material needed to decrypt it.

`async encrypt_empty()`

Encrypt an empty message for the sole purpose of session manangement/ratchet forwarding/key material transportation.

Return type
Tuple[*ContentImpl*, *PlainKeyMaterialImpl*]

Returns
The symmetrically encrypted empty content, and the key material needed to decrypt it.

`async encrypt_key_material(`*session*, *plain_key_material*`)`

Encrypt some key material asymmetrically using the session.

Parameters

- **session** (Session) – The session to encrypt the key material with.
- **plain_key_material** (PlainKeyMaterial) – The key material to encrypt asymmetrically for each recipient.

Return type
EncryptedKeyMaterialImpl

Returns
The encrypted key material.

`async decrypt_plaintext(`*content*, *plain_key_material*`)`

Decrypt some symmetrically encrypted plaintext.

Parameters

- **content** (Content) – The content to decrypt. Not empty, i.e. `Content.empty` will return `False`.
- **plain_key_material** (PlainKeyMaterial) – The key material to decrypt with.

Return type
bytes

Returns
The decrypted plaintext.

Raises
DecryptionFailed – in case of backend-specific failures during decryption.

`async decrypt_key_material(`*session*, *encrypted_key_material*`)`

Decrypt some key material asymmetrically using the session.

Parameters

- **session** (Session) – The session to decrypt the key material with.
- **encrypted_key_material** (EncryptedKeyMaterial) – The encrypted key material.

Return type
PlainKeyMaterialImpl

Returns
The decrypted key material

Raises

- **TooManySkippedMessageKeys** – if the number of message keys skipped by this message exceeds the upper limit enforced by `max_num_per_message_skipped_keys`.
- **DecryptionFailed** – in case of backend-specific failures during decryption.

Warning: Make sure to respect the values of `max_num_per_session_skipped_keys` and `max_num_per_message_skipped_keys`.

Note: When the maximum number of skipped message keys for this session, given by `max_num_per_session_skipped_keys`, is exceeded, old skipped message keys are deleted to make space for new ones.

async signed_pre_key_age()

Return type
`int`

Returns
The age of the signed pre key, i.e. the time elapsed since it was last rotated, in seconds.

async rotate_signed_pre_key()

Rotate the signed pre key. Keep the old signed pre key around for one additional rotation period, i.e. until this method is called again.

Return type
`None`

Returns
Anything, the return value is ignored.

async hide_pre_key(*session*)

Hide a pre key from the bundle returned by *get_bundle()* and pre key count returned by *get_num_visible_pre_keys()*, but keep the pre key for cryptographic operations.

Parameters
session (`Session`) – A session that was passively built using *build_session_passive()*. Use this session to identity the pre key to hide.

Return type
`bool`

Returns
Whether the pre key was hidden. If the pre key doesn't exist (e.g. because it has already been deleted), or was already hidden, do not throw an exception, but return *False* instead.

async delete_pre_key(*session*)

Delete a pre key.

Parameters
session (`Session`) – A session that was passively built using *build_session_passive()*. Use this session to identity the pre key to delete.

Return type
`bool`

Returns
Whether the pre key was deleted. If the pre key doesn't exist (e.g. because it has already been deleted), do not throw an exception, but return *False* instead.

`async delete_hidden_pre_keys()`

Delete all pre keys that were previously hidden using *hide_pre_key()*.

Return type
None

Returns
Anything, the return value is ignored.

`async get_num_visible_pre_keys()`

Return type
int

Returns
The number of visible pre keys available. The number returned here should match the number of pre keys included in the bundle returned by *get_bundle()*.

`async generate_pre_keys(num_pre_keys)`

Generate and store pre keys.

Parameters
num_pre_keys (int) – The number of pre keys to generate.

Return type
None

Returns
Anything, the return value is ignored.

`async get_bundle(bare_jid, device_id)`

Parameters

- **bare_jid** (str) – The bare JID of this XMPP account, to be included in the bundle.
- **device_id** (int) – The id of this device, to be included in the bundle.

Return type
BundleImpl

Returns
The bundle containing public information about the cryptographic state of this backend.

> **Warning:** Do not include pre keys hidden by *hide_pre_key()* in the bundle!

`async purge()`

Remove all data related to this backend from the storage.

Return type
None

Returns
Anything, the return value is ignored.

async purge_bare_jid(*bare_jid*)

Delete all data corresponding to an XMPP account.

Parameters
bare_jid (str) – Delete all data corresponding to this bare JID.

Return type
None

Returns
Anything, the return value is ignored.

class twomemo.twomemo.**AEADImpl**

Bases: AEAD

The AEAD used by this backend as part of the Double Ratchet. While this implementation derives from doubleratchet.recommended.aead_aes_hmac.AEAD, it actually doesn't use any of its code. This is due to a minor difference in the way the associated data is built. The derivation only has symbolic value.

Can only be used with *DoubleRatchetImpl*, due to the reliance on a certain structure of the associated data.

AUTHENTICATION_TAG_TRUNCATED_LENGTH: Final = 16

static _get_hash_function()

Return type
HashFunction

static _get_info()

Return type
bytes

async classmethod encrypt(*plaintext*, *key*, *associated_data*)

Parameters

- **plaintext** (bytes) – The plaintext to encrypt.
- **key** (bytes) – The encryption key.
- **associated_data** (bytes) – Additional data to authenticate without including it in the ciphertext.

Return type
bytes

Returns
The ciphertext.

async classmethod decrypt(*ciphertext*, *key*, *associated_data*)

Parameters

- **ciphertext** (bytes) – The ciphertext to decrypt.
- **key** (bytes) – The decryption key.
- **associated_data** (bytes) – Additional data to authenticate without including it in the ciphertext.

Return type
bytes

Returns
The plaintext.

Raises

- **AuthenticationFailedException** – if the message could not be authenticated using the associated data.
- **DecryptionFailedException** – if the decryption failed for a different reason (e.g. invalid padding).

__annotations__ = {'AUTHENTICATION_TAG_TRUNCATED_LENGTH': 'Final'}

class twomemo.twomemo.**BundleImpl**(*bare_jid*, *device_id*, *bundle*, *signed_pre_key_id*, *pre_key_ids*)

Bases: Bundle

Bundle implementation as a simple storage type.

Parameters

- **bare_jid** (*str*) –
- **device_id** (*int*) –
- **bundle** (*x3dh.Bundle*) –
- **signed_pre_key_id** (*int*) –
- **pre_key_ids** (*Dict[bytes, int]*) –

__init__(*bare_jid*, *device_id*, *bundle*, *signed_pre_key_id*, *pre_key_ids*)

Parameters

- **bare_jid** (str) – The bare JID this bundle belongs to.
- **device_id** (int) – The device id of the specific device this bundle belongs to.
- **bundle** (Bundle) – The bundle to store in this instance.
- **signed_pre_key_id** (int) – The id of the signed pre key referenced in the bundle.
- **pre_key_ids** (Dict[bytes, int]) – A dictionary that maps each pre key referenced in the bundle to its id.

Return type
None

property namespace: str

Return type
str

property bare_jid: str

Return type
str

property device_id: int

Return type
int

property identity_key: bytes

Return type
bytes

__eq__(*other*)

Check an object for equality with this Bundle instance.

Parameters
other (object) – The object to compare to this instance.

Return type
bool

Returns
Whether the other object is a bundle with the same contents as this instance.

Note: The order in which pre keys are included in the bundles does not matter.

__hash__()

Hash this instance in a manner that is consistent with *__eq__()*.

Return type
int

Returns
An integer value representing this instance.

property bundle: Bundle

Returns: The bundle held by this instance.

Return type
Bundle

property signed_pre_key_id: int

Returns: The id of the signed pre key referenced in the bundle.

Return type
int

property pre_key_ids: Dict[bytes, int]

Returns: A dictionary that maps each pre key referenced in the bundle to its id.

Return type
Dict[bytes, int]

class twomemo.twomemo.**ContentImpl**(*ciphertext*)

Bases: Content

Content implementation as a simple storage type.

Parameters
ciphertext (*bytes*) –

__init__(*ciphertext*)

Parameters
ciphertext (bytes) – The ciphertext to store in this instance.

Return type
None

Note: For empty OMEMO messages as per the specification, the ciphertext is set to an empty byte string.

property empty: bool

Returns: Whether this instance corresponds to an empty OMEMO message purely used for protocol stability reasons.

Return type
bool

static make_empty()

Return type
ContentImpl

Returns
An "empty" instance, i.e. one that corresponds to an empty OMEMO message as per the specification. The ciphertext stored in empty instances is a byte string of zero length.

property ciphertext: bytes

Returns: The ciphertext held by this instance.

Return type
bytes

class twomemo.twomemo.**DoubleRatchetImpl**

Bases: DoubleRatchet

The Double Ratchet implementation used by this version of the specification.

MESSAGE_CHAIN_CONSTANT: Final = b'\x02\x01'

static _build_associated_data(*associated_data*, *header*)

Parameters

- **associated_data** (bytes) – The associated data to prepend to the output. If the associated data is not guaranteed to be a parseable byte sequence, a length value should be prepended to ensure that the output is parseable as a unique pair (associated data, header).
- **header** (Header) – The message header to encode in a unique, reversible manner.

Return type
bytes

Returns
A byte sequence encoding the associated data and the header in a unique, reversible way.

__annotations__ = {'MESSAGE_CHAIN_CONSTANT': 'Final', '__aead': 'Type[AEAD]', '__diffie_hellman_ratchet': 'DiffieHellmanRatchet', '__max_num_skipped_message_keys': 'int', '__skipped_message_keys': 'SkippedMessageKeys'}

class twomemo.twomemo.**EncryptedKeyMaterialImpl**(*bare_jid*, *device_id*, *encrypted_message*)

Bases: EncryptedKeyMaterial

EncryptedKeyMaterial implementation as a simple storage type.

Parameters

- **bare_jid** (*str*) –
- **device_id** (*int*) –
- **encrypted_message** (*doubleratchet.EncryptedMessage*) –

__init__(*bare_jid*, *device_id*, *encrypted_message*)

Parameters

- **bare_jid** (str) – The bare JID of the other party.
- **device_id** (int) – The device id of the specific device of the other party.
- **encrypted_message** (EncryptedMessage) – The encrypted Double Ratchet message to store in this instance.

Return type
None

property bare_jid: str

Return type
str

property device_id: int

Return type
int

property encrypted_message: EncryptedMessage

Returns: The encrypted Double Ratchet message held by this instance.

Return type
EncryptedMessage

serialize()

Return type
bytes

Returns
A serialized OMEMOAuthenticatedMessage message structure representing the content of this instance.

static parse(*authenticated_message*, *bare_jid*, *device_id*)

Parameters

- **authenticated_message** (bytes) – A serialized OMEMOAuthenticatedMessage message structure.
- **bare_jid** (str) – The bare JID of the other party.
- **device_id** (int) – The device id of the specific device of the other party.

Return type
EncryptedKeyMaterialImpl

Returns
An instance of this class, parsed from the OMEMOAuthenticatedMessage.

Raises
ValueError – if the data is malformed.

class twomemo.twomemo.**KeyExchangeImpl**(*header*, *signed_pre_key_id*, *pre_key_id*)

Bases: KeyExchange

KeyExchange implementation as a simple storage type.

There are two kinds of instances:

- Completely filled instances
- Partially filled instances received via network

Empty fields are filled with filler values such that the data types and lengths still match expectations.

Parameters

- **header** (*x3dh.Header*) –
- **signed_pre_key_id** (*int*) –
- **pre_key_id** (*int*) –

__init__(*header*, *signed_pre_key_id*, *pre_key_id*)

Parameters

- **header** (Header) – The header to store in this instance.
- **signed_pre_key_id** (int) – The id of the signed pre key referenced in the header.
- **pre_key_id** (int) – The id of the pre key referenced in the header.

Return type
None

property identity_key: bytes

Return type
bytes

builds_same_session(*other*)

Parameters
other (KeyExchange) – The other key exchange instance to compare to this instance.

Return type
bool

Returns
Whether the key exchange information stored in this instance and the key exchange information stored in the other instance would build the same session.

property header: Header

Returns: The header held by this instance.

Return type
Header

property signed_pre_key_id: int

Returns: The id of the signed pre key referenced in the header.

Return type
int

property pre_key_id: int

Returns: The id of the pre key referenced in the header.

Return type
int

is_network_instance()

Return type
bool

Returns
Returns whether this is a network instance. A network instance has all fields filled except for the signed pre key and pre key byte data. The missing byte data can be restored by looking it up from storage using the respective ids.

serialize(*authenticated_message*)

Parameters
authenticated_message (bytes) – The serialized OMEMOAuthenticatedMessage message structure to include with the key exchange information.

Return type
bytes

Returns
A serialized OMEMOKeyExchange message structure representing the content of this instance.

Raises
ValueError – if the serialized OMEMOAuthenticatedMessage is malformed.

static parse(*key_exchange*)

Parameters
key_exchange (bytes) – A serialized OMEMOKeyExchange message structure.

Return type
Tuple[*KeyExchangeImpl*, bytes]

Returns
An instance of this class, parsed from the OMEMOKeyExchange, and the serialized OMEMOAuthenticatedMessage extracted from the OMEMOKeyExchange.

Raises
ValueError – if the data is malformed.

Warning: The OMEMOKeyExchange message structure only contains the ids of the signed pre key and the pre key used for the key exchange, not the full public keys. Since the job of this method is just parsing, the X3DH header is initialized without the public keys here, and the code using instances of this class has to handle the public key lookup from the ids. Use header_filled to check whether the header is filled with the public keys.

class twomemo.twomemo.**MessageChainKDFImpl**

Bases: KDF

The message chain KDF implementation used by this version of the specification.

static _get_hash_function()

Return type
HashFunction

class twomemo.twomemo.**PlainKeyMaterialImpl**(*key*, *auth_tag*)

Bases: PlainKeyMaterial

PlainKeyMaterial implementation as a simple storage type.

Parameters

- **key** (*bytes*) –
- **auth_tag** (*bytes*) –

KEY_LENGTH: Final = 32

__init__(*key*, *auth_tag*)

Parameters

- **key** (bytes) – The key to store in this instance.
- **auth_tag** (bytes) – The authentication tag to store in this instance.

Return type
None

Note: For empty OMEMO messages as per the specification, the key is set to *KEY_LENGTH* zero-bytes, and the auth tag is set to an empty byte string.

property key: bytes

Returns: The key held by this instance.

Return type
bytes

property auth_tag: bytes

Returns: The authentication tag held by this instance.

Return type
bytes

static make_empty()

Return type
PlainKeyMaterialImpl

Returns
An "empty" instance, i.e. one that corresponds to an empty OMEMO message as per the specification. The key stored in empty instances is a byte string of *KEY_LENGTH* zero-bytes, and the auth tag is an empty byte string.

__annotations__ = {'KEY_LENGTH': 'Final'}

class twomemo.twomemo.**RootChainKDFImpl**

Bases: KDF

The root chain KDF implementation used by this version of the specification.

static _get_hash_function()

Return type
HashFunction

static _get_info()

Return type
bytes

class twomemo.twomemo.**SessionImpl**(*bare_jid*, *device_id*, *initiation*, *key_exchange*, *associated_data*, *double_ratchet*, *confirmed=False*)

Bases: Session

Session implementation as a simple storage type.

Parameters

- **bare_jid** (*str*) –
- **device_id** (*int*) –
- **initiation** (*Initiation*) –
- **key_exchange** (KeyExchangeImpl) –
- **associated_data** (*bytes*) –
- **double_ratchet** (DoubleRatchetImpl) –
- **confirmed** (*bool*) –

__init__(*bare_jid*, *device_id*, *initiation*, *key_exchange*, *associated_data*, *double_ratchet*, *confirmed=False*)

Parameters

- **bare_jid** (str) – The bare JID of the other party.
- **device_id** (int) – The device id of the specific device of the other party.
- **initiation** (Initiation) – Whether this session was built through active or passive session initiation.
- **key_exchange** (*KeyExchangeImpl*) – The key exchange information to store in this instance.
- **associated_data** (bytes) – The associated data to store in this instance.
- **double_ratchet** (*DoubleRatchetImpl*) – The Double Ratchet to store in this instance.
- **confirmed** (bool) – Whether the session was confirmed, i.e. whether a message was decrypted after actively initiating the session. Leave this at the default value for passively initiated sessions.

property namespace: str

Return type
str

property bare_jid: str

Return type
str

property device_id: int

Return type
int

property initiation: Initiation

Returns: Whether this session was actively initiated or passively.

Return type
Initiation

property confirmed: bool

In case this session was built through active session initiation, this flag should indicate whether the session initiation has been "confirmed", i.e. at least one message was received and decrypted using this session.

Return type
bool

property key_exchange: *KeyExchangeImpl*

Either the key exchange information received during passive session building, or the key exchange information created as part of active session building. The key exchange information is needed by the protocol for stability reasons, to make sure that all sides can build the session, even if messages are lost or received out of order.

Return type
KeyExchangeImpl

Returns
The key exchange information associated with this session.

property receiving_chain_length: Optional[int]

Returns: The length of the receiving chain, if it exists, used for own staleness detection.

Return type
Optional[int]

property sending_chain_length: int

Returns: The length of the sending chain, used for staleness detection of other devices.

Return type
int

property associated_data: bytes

Returns: The associated data held by this instance.

Return type
bytes

property double_ratchet: *DoubleRatchetImpl*

Returns: The Double Ratchet held by this instance.

Return type
DoubleRatchetImpl

confirm()

Mark this session as confirmed.

Return type
None

class twomemo.twomemo.**StateImpl**

Bases: BaseState

The X3DH state implementation used by this version of the specification.

INFO: Final = b'OMEMO X3DH'

IDENTITY_KEY_ENCODING_LENGTH: Final = 32

static **_encode_public_key**(*key_format*, *pub*)

Parameters

- **key_format** (IdentityKeyFormat) – The format in which this public key is serialized.
- **pub** (bytes) – The public key.

Return type
bytes

Returns
An encoding of the public key, possibly including information about the curve and type of key, though this is application defined. Note that two different public keys must never result in the same byte sequence, uniqueness of the public keys must be preserved.

__annotations__ = {'IDENTITY_KEY_ENCODING_LENGTH': 'Final', 'INFO': 'Final', '__hash_function': 'HashFunction', '__hidden_pre_keys': 'Set[PreKeyPair]', '__identity_key': 'IdentityKeyPair', '__identity_key_format': 'IdentityKeyFormat', '__info': 'bytes', '__old_signed_pre_key': 'Optional[SignedPreKeyPair]', '__pre_keys': 'Set[PreKeyPair]', '__signed_pre_key': 'SignedPreKeyPair'}

PYTHON MODULE INDEX

t

twomemo.twomemo, 7

INDEX

Symbols

__annotations__ (*twomemo.twomemo.AEADImpl attribute*), 14
__annotations__ (*twomemo.twomemo.DoubleRatchetImpl attribute*), 16
__annotations__ (*twomemo.twomemo.PlainKeyMaterialImpl attribute*), 20
__annotations__ (*twomemo.twomemo.StateImpl attribute*), 23
__eq__() (*twomemo.twomemo.BundleImpl method*), 15
__hash__() (*twomemo.twomemo.BundleImpl method*), 15
__init__() (*twomemo.twomemo.BundleImpl method*), 14
__init__() (*twomemo.twomemo.ContentImpl method*), 15
__init__() (*twomemo.twomemo.EncryptedKeyMaterialImpl method*), 17
__init__() (*twomemo.twomemo.KeyExchangeImpl method*), 18
__init__() (*twomemo.twomemo.PlainKeyMaterialImpl method*), 20
__init__() (*twomemo.twomemo.SessionImpl method*), 21
__init__() (*twomemo.twomemo.Twomemo method*), 7
_build_associated_data() (*twomemo.twomemo.DoubleRatchetImpl static method*), 16
_encode_public_key() (*twomemo.twomemo.StateImpl static method*), 23
_get_hash_function() (*twomemo.twomemo.AEADImpl static method*), 13
_get_hash_function() (*twomemo.twomemo.MessageChainKDFImpl static method*), 19
_get_hash_function() (*twomemo.twomemo.RootChainKDFImpl static method*), 20
_get_info() (*twomemo.twomemo.AEADImpl static method*), 13
_get_info() (*twomemo.twomemo.RootChainKDFImpl static method*), 21

A

AEADImpl (*class in twomemo.twomemo*), 13
associated_data (*twomemo.twomemo.SessionImpl property*), 22
auth_tag (*twomemo.twomemo.PlainKeyMaterialImpl property*), 20
AUTHENTICATION_TAG_TRUNCATED_LENGTH (*twomemo.twomemo.AEADImpl attribute*), 13

B

bare_jid (*twomemo.twomemo.BundleImpl property*), 14
bare_jid (*twomemo.twomemo.EncryptedKeyMaterialImpl property*), 17
bare_jid (*twomemo.twomemo.SessionImpl property*), 21
build_session_active() (*twomemo.twomemo.Twomemo method*), 8
build_session_passive() (*twomemo.twomemo.Twomemo method*), 9
builds_same_session() (*twomemo.twomemo.KeyExchangeImpl method*), 18
bundle (*twomemo.twomemo.BundleImpl property*), 15
BundleImpl (*class in twomemo.twomemo*), 14

C

ciphertext (*twomemo.twomemo.ContentImpl property*), 16
confirm() (*twomemo.twomemo.SessionImpl method*), 22
confirmed (*twomemo.twomemo.SessionImpl property*), 22
ContentImpl (*class in twomemo.twomemo*), 15

D

decrypt() (*twomemo.twomemo.AEADImpl class method*), 13

decrypt_key_material() (*twomemo.twomemo.Twomemo method*), 10
decrypt_plaintext() (*twomemo.twomemo.Twomemo method*), 10
delete_hidden_pre_keys() (*twomemo.twomemo.Twomemo method*), 12
delete_pre_key() (*twomemo.twomemo.Twomemo method*), 11
device_id (*twomemo.twomemo.BundleImpl property*), 14
device_id (*twomemo.twomemo.EncryptedKeyMaterialImpl property*), 17
device_id (*twomemo.twomemo.SessionImpl property*), 21
double_ratchet (*twomemo.twomemo.SessionImpl property*), 22
DoubleRatchetImpl (*class in twomemo.twomemo*), 16

E

empty (*twomemo.twomemo.ContentImpl property*), 16
encrypt() (*twomemo.twomemo.AEADImpl class method*), 13
encrypt_empty() (*twomemo.twomemo.Twomemo method*), 10
encrypt_key_material() (*twomemo.twomemo.Twomemo method*), 10
encrypt_plaintext() (*twomemo.twomemo.Twomemo method*), 9
encrypted_message (*twomemo.twomemo.EncryptedKeyMaterialImpl property*), 17
EncryptedKeyMaterialImpl (*class in twomemo.twomemo*), 16

G

generate_pre_keys() (*twomemo.twomemo.Twomemo method*), 12
get_bundle() (*twomemo.twomemo.Twomemo method*), 12
get_num_visible_pre_keys() (*twomemo.twomemo.Twomemo method*), 12

H

header (*twomemo.twomemo.KeyExchangeImpl property*), 18
hide_pre_key() (*twomemo.twomemo.Twomemo method*), 11

I

identity_key (*twomemo.twomemo.BundleImpl property*), 14
identity_key (*twomemo.twomemo.KeyExchangeImpl property*), 18
IDENTITY_KEY_ENCODING_LENGTH (*twomemo.twomemo.StateImpl attribute*), 23
INFO (*twomemo.twomemo.StateImpl attribute*), 23
initiation (*twomemo.twomemo.SessionImpl property*), 22
is_network_instance() (*twomemo.twomemo.KeyExchangeImpl method*), 19

K

key (*twomemo.twomemo.PlainKeyMaterialImpl property*), 20
key_exchange (*twomemo.twomemo.SessionImpl property*), 22
KEY_LENGTH (*twomemo.twomemo.PlainKeyMaterialImpl attribute*), 20
KeyExchangeImpl (*class in twomemo.twomemo*), 17

L

load_session() (*twomemo.twomemo.Twomemo method*), 8

M

make_empty() (*twomemo.twomemo.ContentImpl static method*), 16
make_empty() (*twomemo.twomemo.PlainKeyMaterialImpl static method*), 20
MESSAGE_CHAIN_CONSTANT (*twomemo.twomemo.DoubleRatchetImpl attribute*), 16
MessageChainKDFImpl (*class in twomemo.twomemo*), 19
module
    twomemo.twomemo, 7

N

namespace (*twomemo.twomemo.BundleImpl property*), 14
namespace (*twomemo.twomemo.SessionImpl property*), 21
namespace (*twomemo.twomemo.Twomemo property*), 7

P

parse() (*twomemo.twomemo.EncryptedKeyMaterialImpl static method*), 17
parse() (*twomemo.twomemo.KeyExchangeImpl static method*), 19
PlainKeyMaterialImpl (*class in twomemo.twomemo*), 20
pre_key_id (*twomemo.twomemo.KeyExchangeImpl property*), 18
pre_key_ids (*twomemo.twomemo.BundleImpl property*), 15

purge() (*twomemo.twomemo.Twomemo method*), 12
purge_bare_jid() (*twomemo.twomemo.Twomemo method*), 12

R

receiving_chain_length (*twomemo.twomemo.SessionImpl property*), 22
RootChainKDFImpl (*class in twomemo.twomemo*), 20
rotate_signed_pre_key() (*twomemo.twomemo.Twomemo method*), 11

S

sending_chain_length (*twomemo.twomemo.SessionImpl property*), 22
serialize() (*twomemo.twomemo.EncryptedKeyMaterialImpl method*), 17
serialize() (*twomemo.twomemo.KeyExchangeImpl method*), 19
SessionImpl (*class in twomemo.twomemo*), 21
signed_pre_key_age() (*twomemo.twomemo.Twomemo method*), 11
signed_pre_key_id (*twomemo.twomemo.BundleImpl property*), 15
signed_pre_key_id (*twomemo.twomemo.KeyExchangeImpl property*), 18
StateImpl (*class in twomemo.twomemo*), 22
store_session() (*twomemo.twomemo.Twomemo method*), 8

T

Twomemo (*class in twomemo.twomemo*), 7
twomemo.twomemo
    module, 7